VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Přepracování počítačové hry Hobo: Tough Life do virtuální reality

Update of Hobo: Tough Life Game of Virtual Reality Experience

2018 Jiří Vašica

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Vašica**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Přepracování počítačové hry Hobo: Tough Life do virtuální reality
Update of Hobo: Tough Life Game of Virtual Reality Experience

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem této práce je vytvořit počítačovou hru pro virtuální realitu SteamVR, která bude vycházet z předlohy Hobo: Tough Life. V rámci bakalářské práce student prostuduje možnosti a omezení virtuální reality oproti klasickým hrám a pomocí těchto znalostí následně uzpůsobí předlohu současným možnostem virtuální reality.

Struktura práce bude následující:
1. Průzkum a zhodnocení současného stavu virtuální reality.
2. Návrh a úprava designu jednotlivých částí desktopové hry možnostem virtuální reality.
3. Tvorba hry a provedení funkčních testů s HTC VIVE na operačním systému Windows.
4. Zhodnocení výsledku v závěru.

Seznam doporučené odborné literatury:

[1] Projekt OpenVR, https://github.com/ValveSoftware/openvr

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2018 ..... Vaška .....

Rád bych na tomto místě poděkoval svému vedoucímu panu Ing. Janu Gaurovi, Ph.D. za rady a konzultace při vytváření této práce.

Abstrakt

Práce se zabývá návrhem a vývojem her pro virtuální realitu na platformě SteamVR za pomocí herního enginu Unity. Na základě herní předlohy s názvem Hobo: Tough Life je vyvíjena hra, která má přizpůsobeno ovládání, grafickou stylizaci a herní mechaniky z této hry pro prostředí virtuální reality vzhledem k současným technologickým možnostem a omezením, která jsou spojená s aktuálně dostupným hardwarem pro virtuální realitu.

Klíčová slova: virtuální realita, počítačová hra, Unity, grafika

Abstract

The thesis researches design of virtual reality games and also describes the development of games for virtual reality on SteamVR using game engine Unity. The game for virtual reality is developed based on the computer game called Hobo: Tough Life with customized controls, graphical stylization and game mechanics for this platform based on current possibilities and limitations of the latest available headsets.

Key Words: virtual reality, computer game, Unity, graphics

Obsah

Seznam použitých zkratek a symbolů

2D	–	Two-dimensional
3D	–	Three-dimensional
AI	–	Artificial Intelligence
RPG	–	Role Playing Game
NPC	–	Non-Player Character
GUI	–	Graphical User Interface
PBR	–	Physically Based Rendering
JSON	–	JavaScript Object Notation
SPSR	–	Single Pass Stereo Rendering
SPISR	–	Single Pass Instanced Stereo Rendering
API	–	Application Programming Interface
DDR	–	Dynamic random-access memory
SDK	–	Software development kit
RAM	–	Random-access memory
GPU	–	Graphic processing unit
CPU	–	Central processing unit
FPS	–	Frames per second
HUD	–	Heads-up Display
LOD	–	Level of Detail
HD	–	High-definition
VR	–	Virtual Reality
GB	–	Gigabyte
Hz	–	Hertz

Seznam obrázků

Seznam tabulek

1 Úvod

Tato bakalářská práce se zaměřuje na oblast označovanou jako virtuální realita (VR). Technologie pro VR je stále ještě relativně v raném stádiu, avšak jedná se o velice rychle rostoucí odvětví s obrovským potenciálem napříč velkým spektrem odlišných oborů, kde se v následujících letech pro tuto technologii očekává exponenciální růst v počtu aktivně používaných zařízení a dynamický rozvoj [1, 2].

Cílem této práce je vytvořit hru pro virtuální realitu běžící na platformě SteamVR na základě herní předlohy Hobo: Tough Life a zdokumentovat, co takový proces obnáší. Tato práce nejprve v kapitole 2 popisuje, co vlastně virtuální realita je a jaké jsou její možnosti využití a problémy. Specifikuje, s čím je nutno počítat a na co brát ohled při vývoji pro VR, na rozdíl od běžného vývoje počítačových (a konzolových) her. V kapitole 3 se tato práce zabývá návrhem a designem samotné hry a jednotlivých herních mechanik v ní na základě předlohy Hobo: Tough Life vzhledem k současným technickým možnostem virtuální reality. Technickou realizací jednotlivých designových návrhů a celkovým technickým zpracováním (například práce s grafikou, chování umělé inteligence, práce se zvukem atd.) se zaobírá kapitola 4. Průběh testování během vývoje za pomocí HTC Vive na platformě SteamVR je popsán v kapitole 5. Zhodnocení výsledného přepracování podle počítačové předlohy a jaké poznatky přinesl tento vývoj jsou obsaženy v závěrečné kapitole 6.

2 Virtuální realita

Pojmem virtuální realita se označuje zobrazení simulovaného prostředí v prostoru okolo uživatele. Snahou VR je tedy vytvoření iluze, že se člověk v tomto fiktivním prostoru opravdu nachází. Existuje celá řada odvětví využívající tuto technologii. Jedná se např. o simulování operací ve zdravotnictví, prostorová vizualizace výsledného architektonického díla ve stavebnictví, armádní simulace, demonstrativní ukázky ve školství nebo využití v zábavním průmyslu jako jsou hry, filmy a mnoho dalších [3, 4].

2.1 Současný stav

Samotná myšlenka zobrazování umělého prostředí existuje již dlouho. Ačkoliv první pokusy o vytvoření techniky simulující virtuální realitu sahají až do začátku 50. let 20. století, teprve před pár lety pokročil vývoj v této oblasti natolik, aby umožnil vznik prvních brýlí pro virtuální realitu, které nabízejí věrohodný a pohodlný zážitek pro hraní her ve VR. Mezi současné nejpopulárnější a zároveň i nejkvalitnější brýle na trhu pro VR se řadí Oculus Rift a HTC Vive. Obě tato zařízení byla vydána v roce 2016. Novější vývojové verze těchto zařízení se plánují na druhou polovinu roku 2018 [5, 6].

2.2 Problémy

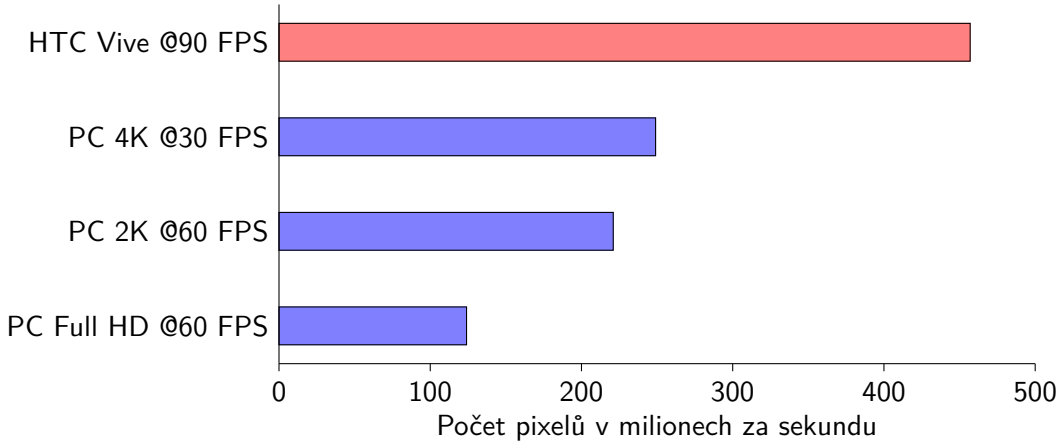
I přes výrazný posun od prvních prototypů se současné brýle stále potýkají s řadou zásadních technických problémů a omezení, se kterými je třeba při vývoji a návrhu neustále počítat. Mezi nejčastější problémy se řadí:

- velké nároky na výkon,
- kinetóza,
- nízké rozlišení,
- šum,
- nízká herní doba,
- cena,
- zdravotní problémy.

2.2.1 Nároky na výkon

V herním průmyslu se stále nejčastěji při vykreslování v reálném čase používá Full HD rozlišení (High-definition, 1920×1080) při 60 snímcích za vteřinu (zkratka FPS, anglicky frames per second), resp. 30 FPS u herních konzolí. Při Full HD se jedná o zhruba 124 milionů pixelů, které

je třeba zpracovat za pomocí Graphic Processing Unit (GPU). VR na rozdíl od běžného Full HD vykreslování při 60 FPS vyžaduje skokový nárust v požadavcích na výkon. Pro eliminování negativních účinků na člověka (viz sekce 2.2.2) a snížení odezvy mezi akcí uživatele a zobrazením výsledku na displeji na minimum je třeba přiblížit zobrazovací technologie v brýlích pro VR co nejvíce lidskému oku, což znamená velmi vysoké rozlišení a rychlou obnovovací frekvenci s co nejnižší odezvou. Současné nejlepší brýle na trhu pro VR disponují stereoskopickým frame bufferem o rozlišení 2160×1200 (1080×1200 na jedno oko) s obnovovací frekvencí 90 hertzů (Hz). Toto rozlišení je používáno při konečném zobrazení na panelech v brýlích, při vykreslování u VR však potřebujeme renderovat do lehce většího rozlišení, než jaké je k dispozici v brýlích. Jedná se o takzvaný off-screen rendering, který u současných VR zařízeních znamená zhruba 1,4násobek v každém směru. Což při rozlišení 1080×1200 na jedno oko činí 1512×1680. Dostáváme se tedy v hrubých číslech na 457 milionu pixelů za vteřinu, což je nárůst o téměř 380% oproti běžnému renderování ve Full HD. Na obrázku č. 1 lze vidět názorněji tento nárůst nároků u HTC Vive vzhledem k běžnému vykreslování u počítačových her [7].



Obrázek 1: Srovnání nároků na vykreslování

2.2.2 Kinetóza

Kinetóza neboli nemoc z pohybu je jedním z nejčastějších problémů ve VR. Tento problém nastává při nesrovnalostech mezi pohybem v realitě a pohybem ve VR (např. když dochází k posunutí ve virtuálním prostoru, zatímco tělo v reálném prostředí stojí na místě). Mozek si pak totiž myslí, že jsme byli otráveni a snaží se tedy jedu zbavit (např. zvracením). Mezi nejčastější příznaky patří bolest hlavy, nevolnost, zvracení, bledost, pocení, dezorientace a únava. Každý jedinec tento problém vnímá jinak [8, 9].

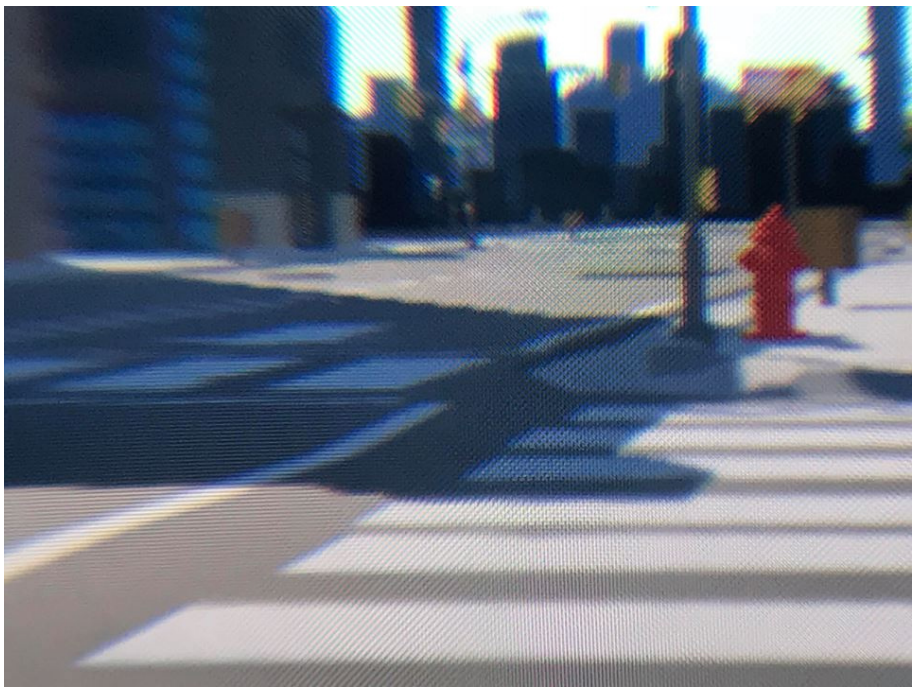
2.2.3 Rozlišení

I když rozlišení v současných VR brýlích je větší než rozlišení ve většině běžně dostupných televizích, monitorech atd., tak v porovnání s rozlišením lidského oka se stále jedná o rozlišení velmi

nízké. Pro dosažení stejné hustoty pixelů, jako když se díváme na monitor v běžné vzdálenosti, bychom museli mít na jedno oko displej o rozlišení 5K × 5K. Pokud bychom chtěli dosáhnout hustoty sítnice lidského oka, pak by bylo třeba mít až 16K × 16K rozlišení na jedno oko [10]. Tento fakt má za následek, že při současné technologii se text či různé drobné detaily stavají problémem. I s pomocí metod jako supersampling je drobnější text značně rozmazaný a málo čitelný. Tento jev sám o sobě nemá nějak zásadní vliv na samotný zážitek z VR, ale na druhou stranu je limitující pro vývojáře v tom, co mohou uživatelům na obrazovce zobrazit.

2.2.4 Viditelnost pixelové mřížky

Problém viditelnosti pixelové mřížky (v angličtině screen door effect, ukázka viz obrázek č. 2) je zobrazovací artefakt displejů, kde lze lidským okem pozorovat jednotlivé předěly oddělující od sebe pixely na obrazovce. Tento problém se při běžné práci s monitorem či mobilním telefonem pozorovat nedá, jelikož tato zařízení používáme v takové vzdálenosti od očí, při níž jsou předěly pixelů pro náš mozek zanedbatelně malé, a při výsledném zpracování obrazu pak tyto artefakty nejsou vidět. U brýlí pro VR se displej nachází v těsné blízkosti očí, přičemž při rozlišení nižším než rozlišení lidského oka jsou následně tyto předěly lidským okem pozorovatelné [11].



Obrázek 2: Ukázka problému viditelnosti pixelové mřížky

2.2.5 Herní doba

Na rozdíl od běžných her pro herní konzole a počítače, jejichž hraním můžeme trávit velké množství času (v řádech hodin), se herní doba ve VR při použití současného hardwaru pohybuje

v řádech minut. Kratší herní doba je dána poměrně velkou váhou současných VR brýlí, které značně namáhají naše krční svaly. Mezi další důvody se pak řadí únava očí či celková fyzická vyčerpanost (jelikož ve VR vynakládáme větší množství pohybu než při běžném hraní v sedě s klávesnicí u počítače) [12]. I když ve VR hře nebo aplikaci minimalizujeme či zcela eliminujeme výskyt situací způsobující kinetózu, tak i přesto se po delším hraní začínají projevovat stavy určitého vnitřního nepohodlí.

2.2.6 Cena

Jako většina nových technologií jsou VR brýle v současnosti drahé, pro spoustu lidí je tak cena hlavní překážkou jejich pořízení. Nezbytný je i dostatečně výkonný počítač, aby bylo vůbec možné brýle používat. Současně se tedy nejedná o mainstreamovou záležitost. Tento problém může být časem vyřešen zdokonalováním výrobních postupů a snižováním výrobních nákladů. Na trh se tak budou moci dostat kvalitnější a cenově dostupnější zařízení. Tento trend můžeme již pozorovat na současně nejprodávanějších brýlích pro VR (HTC Vive a Oculus Rift), jejichž pořizovací cena byla od vstupu na trh snížena až o polovinu oproti původní maloobchodní ceně [13].

2.2.7 Zdravotní problémy

Jak již bylo popsáno v sekci 2.2.2, v určitých situacích se může lidem ve VR udělat špatně. Avšak jedná se pouze o dočasnou nevolnost, která po určité době odezní. Stále se však neví, zdali používání VR brýlí má na člověka nějaké (a případně jaké) trvalé negativní účinky. Podle vědců je na zjištění dlouhodobých důsledků VR potřeba nasbírat více dat a až časem se teprve ukáže, jestli (a jak moc) je používání VR brýlí škodlivé pro náš organismus [14].

3 Návrh

Tato kapitola nejprve stanovuje základní zásady, na něž je potřeba při designu pamatovat, a následně se zabývá rozborem designu herní předlohy vybráné pro tuto práci a jeho přepracováním do podoby vhodné pro VR.

Při úpravě herního návrhu se nebudeme snažit o převedení celé hry do VR jedna ku jedné, neboť v rámci rozsahu této práce pro celkové přepracování nemáme dostatek prostoru a rovněž, jak si v následujících podkapitolách ukážeme, některé prvky se do VR nedají převést bez alternativního řešení.

3.1 Základní principy

Se snahou o maximální využití výhod platformy VR a minimalizování negativních dopadů popsaných v kapitole 2.2 byly stanoveny následující principy, na které musíme při návrhu brát zřetel:

- plynulost hry a vnitřní pohodlí uživatele,
- přenesení lokomoce hráče z reality do VR,
- veškerý pohyb ve VR řízený samotným hráčem,
- design zaměřený na intenzivní, avšak krátký zážitek,
- komplexnost úkonů a dílčích mechanik přizpůsobena herní době,
- přímá interakce s virtuálním světem,
- nenáročnost na oči (vyhýbání se drobným textům či nutnost ostřit na předměty v dálce),
- využití fyzické přítomnosti uživatele v herním světě.

3.2 Analýza předlohy a návrh herního designu

Jako předloha pro tuto práci byla vybrána počítačová hra Hobo: Tough Life. Žánrově se jedná o hru na hrdiny (zkratka RPG, anglicky Role-Playing Game) odehrávající se ve trojdimenzionálním (3D) otevřeném městském prostředí, kde se hráč staví do role bezdomovce. Hlavním cílem v této hře je snaha o přežití. Tato hra není cílená pro každého hráče, jelikož se profiluje jako velmi těžká [15].

Na základě prostudování hry byly stěžejní mechaniky předlohy definovány následujícím způsobem:

- městské prostředí,
- RPG prvky - parametry postavy (zdraví, hlad, únava atd.),

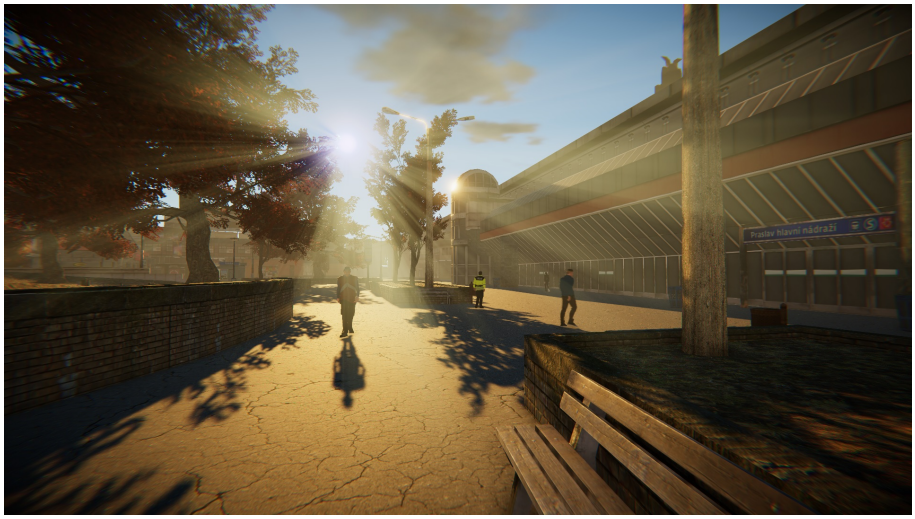
- inventář,
- vybírání popelnic,
- ohřívání u barelu,
- interakce s lidmi,
- obchody,
- počasí.

3.2.1 Městské prostředí

Hra Hobo: Tough Life se odehrává v městském prostředí, kde je kladen velký důraz na variabilitu. Nachází se zde desítky typů budov. Lze zde nalézt rozsáhla sídliště, hustě zalidněné oblasti, odlehlá místa nebo třeba parky. Napříč herní mapou se mezi jednotlivými oblastmi často vyskytují výškové rozdíly, a hráč tak opakovaně musí chodit do kopce, překonávat schody, prohlubně atd. Tento vertikální level design herního světa je však pro VR problém. Při používání VR brýlí se totiž hráč pohybuje v jedné rovině. Pokud by například nastala situace, kdy se hráč v reálném prostředí pohybuje směrem rovně při stále stejné výšce, ale ve VR se jeho virtuální postava pohybuje do kopce (směrem rovně, ale zároveň mírně nahoru), dochází ke kolizi vnímaní lokomoce. To by mohlo vést až k vyvolání kinetózy. Je tedy zřejmé, že u level designu města bude nezbytné, aby veškeré herní prostředí, ve kterém se hráč může pohybovat, obsahovalo minimum výškových rozdílů.

Grafická stylizace městského prostředí i veškerá další grafika má v Hobo: Tough Life realistickou podobu (viz obrázek č. 3). Používají se zde textury vyrobené ze skutečných fotek, které jsou zpracovány do velmi vysokého rozlišení (až 16K). Pro dotvoření realistického vzhledu jsou ve hře použity náročné, zato věrohodně vypadající Physically Based Rendering (PBR) shadery, volumetrické osvětlení, dynamický skybox či značné množství postprocessingových efektů (temporální antialiasing, screen space reflection, ambient occlusion, tonemapping, motion blur, bloom, eye adaptation atd.). Na nasvícení herního světa se používá dynamické světlo. Kromě běžného směrového osvětlení se navíc projevuje v reálném čase takzvané předpočítané nepřímé osvětlení (globální iluminace). I přes velké množství optimalizačních technik (streamování textur, level of detail - LOD, atlasování textur atd.) je hra velmi náročná jak na Central Processing Unit (CPU), tak na GPU. Mezi jeden z největších problémů se řadí rychlost mapovaní textur na dílčí pixely (v angličtině pod označením fillrate), která je dána počtem pixelů obrazovky $\times$ složitost shaderů $\times$ počet překreslení [16]. Jak již zaznělo v sekci 2.2.1, současné brýle pro virtuální realitu zvedají množství vykreslovaných pixelů o 380% oproti Full HD, což v situaci, kdy již předloha má problémy s tokem pixelů při Full HD, značí, že pro potřeby této práce bude nutné pracovat s grafickým vizuálem odlišným způsobem.

Nabízejí se dvě možná řešení daného problému. V první variantě se zachová snaha o realistický grafický styl, přičemž se razantně sníží rozlišení jednotlivých textur, kompletně se odstraní postprocessingové efekty, dojde ke zjednodušení shaderů atd., výsledná kvalita však půjde znatelně dolů (např. rozmazané textury). Ve druhé variantě lze využít nízko polygonální grafický styl, jenž se vzdaluje od realistického zobrazení a má blíže spíše ke komiksové stylizaci. Hlavní rozpoznávacím znakem polygonální grafiky je používání modelů s nízkým počtem polygonů, které využívají takzvané ploché stínování. V tomto grafickém stylu se nesnažíme o co nejhladší přechody mezi dílčími plochami, ale naopak je naším cílem jednotlivé hrany modelu zvýraznit. Podobnou filozofii jednoduchosti lze aplikovat i na textury, kde se místo skladby z reálných fotek používají textury obsahující pouze základní barvy či jednoduché vzory a obrázky. Za pomoci tohoto přístupu stačí ve hře využití malého množství textur při nízkém rozlišení, aniž by výsledná grafická podoba působila jakkoliv deformovaně [17]. Pro tuto práci byl zvolen nízko polygonální grafický styl z důvodu razantně nižších nároků na výkon a možnosti dosáhnout kvalitní vizualizace bez viditelných kompromisů.



Obrázek 3: Herní předloha Hobo: Tough Life

3.2.2 RPG prvky

Hra Hobo: Tough Life je zástupcem survival RPG žánru. Definice konkrétních prvků pro kategorii RPG se různí. U her je zcela běžné kombinovat herní žánry dohromady. Ty pak v dané kombinaci tvoří své vlastní subkategorie [18]. Za obecné prvky a znaky RPG lze považovat následující:

- hráč zastává roli fiktivního postavy (někdy i vícero postav), se kterou následně prožívá její život v rámci virtuálního světa dané počítačové hry,
- herní postava je tvořena vlastnostmi a potřebami této postavy, se kterými hráč při hraní musí počítat a kooperovat,

- cílem hry je splnění předem definovaného centrálního příběhu či úkolu, jenž hráče provází po celou dobu hraní [18].

Přívlastek survival v žánru předlohy znamená, že hlavním úkolem je v tomto případě přežit, a to pokud možno co nejdéle. Přežívání je ve hře abstraktně znázorněno za pomoci parametru reprezentujícího aktuální zdraví postavy. Aby tento parametr měl smysl, musí ve hře existovat rovněž určitý druh nebezpečí, které je vytvořeno s cílem redukovat zdraví hráčovy postavy až do bodu, kdy tento virtuální charakter umírá [19].

Ve hře Hobo: Tough Life je stav postavy bezdomovce ovlivňován velkým množstvím parametrů. Některé z nich jsou dokonce hráči skryty a fungují na pozadí. Mezi primární parametry patří hráčovo zdraví, úroveň hladu, současná morálka, únava a úroveň prochladnutí a promoknutí. K sekundárním parametrům se řadí například aktuální zápach či výdrž (stamina). Ve hře se dále nacházejí také takzvané vedlejší parametry, kam patří například celková imunita, odolnost proti prochladnutí atd. Jedná se však pouze o podpůrné veličiny, kterými jsou ovlivňovány primární a sekundární parametry.

Jednotlivé parametry jsou ve hře reprezentovány jako interval (který například u zdraví nabývá hodnot od 0 do 100). Pro primární a sekundární parametry jsou navíc ještě definovány i takzvané stavy. Tyto stavy se aktivují od určité hranice na daném intervalu jednotlivých parametrů, přičemž každý stav mění působení dané veličiny na hráče. Například parametr prochladnutí je definován v intervalu od 0 do 100. Jeho výchozí stav značí, že hráč je v teple, a prochladnutí se tedy na hráči negativně neprojevuje. Jakmile však úroveň prochladnutí klesne na své stupnici pod hodnotu 20, pak pro tuto veličinu nastává stav promrzlý, který průběžně snižuje hráčovo zdraví.

Pro přehlednost se všechny parametry a aktuální stavy postavy nacházejí ve virtuálním diáři. Tento diář neslouží pouze pro vizualizaci parametrů, ale používá se i pro znázornění inventáře, obsahuje přehled dosažených úrovní dílčích dovedností atd. Z obrázku č. 4 je patrné, že diář je zpracovaný v podobě dvoudimenzionálního grafického uživatelské rozhraní (2D GUI). Pro omezení nutnosti neustálého zobrazování diáře se nejdůležitější parametry a stavy zobrazují v Heads-up displeji (HUD) na hlavní obrazovce. Stejně jako diář je i HUD realizovaný formou 2D GUI. Použití takového způsobu vizualizace je však ve VR nemyslitelné (viz kapitoly 2.2.3 a 3.1), a tedy jak diář, tak i HUD či jakýkoliv jiný 2D prvek se ve hře použít nedá. Ztrácí se tak možnost zobrazit hráči pohromadě všechny parametry s potřebnými vlastnostmi a přehled jednotlivých stavů postavy. Je tudíž třeba vymyslet jiný způsob, jak dát hráči jednoznačně, rychle a intuitivně najevo, jak na tom momentálně jeho postava je.

Kromě vizualizace jednotlivých veličin však narážíme i na další problém. Konkrétně se týká množství jednotlivých parametrů a stavů. Jak již bylo stanoveno v základních principech (viz kapitola 3.1), snahou není (na rozdíl od předlohy a obecně počítačových her) dlouhá herní doba, ale spíše kratší, avšak intenzivní prožitek. Při zkrácené hratelnosti by tolik parametrů pohromadě jako v Hobo: Tough Life nikdy nemohlo fungovat. Jednak neexistuje způsob, jak takové množství parametrů najednou znázornit, a navíc jakákoliv aktivita ve VR je fyzicky

Obrázek 4: Vizualizace stavů a parametrů v Hobo: Tough Life

daleko náročnější než běžné hraní v sedě. Spravování a udržování velkého množství veličin by vedlo ke dvěma možnostem. Buď by působnost jednotlivých parametrů bylo nezbytné zredukovat natolik, že by se ze hry vytratila veškerá dynamika a intenzita, a nebo by jejich působení bylo znát, avšak hráč by následně ve hře nevěděl, co dřív udělat, a rychle by se tak u něj dostavila frustrace. Pro řešení tohoto problému se uchýlíme k redukování parametrů pro hráče ve VR na zvladatelný počet.

Ve VR hráč zažívá skutečnou fyzickou únavu a nedostatek energie, tudíž vlastnosti jako stamina a únava lze rovnou vypustit, jelikož by bylo zbytečné mít tyto prvky zakomponované ve hře dvakrát. Parametr morálky se používá pro zachycení situace, kdy se hráči něco nepovede nebo když se ve hře uchýlí k něčemu nemorálnímu. Zde se opět můžeme spolehnout na větší imerzi díky VR a místo herního parametru využijeme přímo fyzickou morálku daného hráče. Kvalitně zpracovaný zážitek ve VR, kde hráč je přímo uprostřed dění, se vždy daleko více projeví na opravdové morálce, než když se obdobná věc stane na monitoru. Úroveň prochladnutí a promoknutí spolu úzce souvisí. Promoknutí nabývá na hodnotě, pokud se hráč vystaví dešti. Při silném promoknutí dochází k ovlivnění úrovně prochladnutí, která je navíc ještě ovlivněna venkovní teplotou. V rámci snahy o zjednodušování se tedy přikloníme ke spojení těchto dvou veličin v jednu, kdy teplota i srážky budou ovlivňovat identický parametr, avšak s rozdílnou intenzitou působnosti. Sekundární vlastnosti úzce souvisí s primárními a přímo (někdy i nepřímo) se podílejí na jejich ovlivňování. Toto se však nedá aplikovat na parametr zápachu. Ten na rozdíl od ostatních sekundárních parametrů nepůsobí na postavu, ale ovlivňuje chování lidí v okolí hráče. Jelikož ostatní sekundární parametry slouží hlavně k umocnění těch primárních a samy o sobě nepřinášejí unikátní chování, omezíme se v této kategorii výhradně na zápach.

3.2.2.1 Vizualizace parametrů ve VR Po simplifikaci parametrů se jejich seznam zúžil na život, hlad, prochladnutí a zápach. Šest primárních veličin bylo redukováno na tři a z

pěti sekundárních zbyla jedna. Komplexnost tak byla uzpůsobena herní době ve VR. Nyní je třeba navrhnout způsob vizualizace těchto parametrů. Podíváme-li se na jednotlivé parametry, je zřejmé, že se jedná o běžné životní situae. V realitě člověk pomocí smyslů snadno rozpozná, zda je mu zima, jak na tom je zdravotně atd. Současná technologie pro virtuální realitu umožňuje tři z pěti našich smyslů evokovat. Zrak je hráči zprostředkován pomocí displeje ve VR brýlích, zvukové vjemy se k němu dostávají přes sluchátka a hmat lze simulovat pomocí haptiky zabudované v ovladačích. Tento poznatek se tedy využije u následujícího designu.

O prochladnutí našeho těla se dozvídáme pomocí nervů v těle. Tento stav jen pomocí haptiky v ovladačích není možné v současnosti do VR přenést. Promrznutí se však dá rozpoznat i pomocí jiného smyslu než hmatu, konkrétně zrakem. Při nízkých teplotách lidská kůže lehce modrá, čímž se dá inspirovat. Při podchlazení hráče tak bude docházet ke zbarvení kůže virtuální postavy.

Dalším parametrem je hlad. Když se v reálném světě dostaví pocit hladu, je tento stav doprovázen zvuky, které vydává lidské střevo při pohybu plynů a tekutin [20]. Jelikož ve VR je zvuk zastoupen pomocí sluchátek, tak nám pro evokování hladu stačí přehrávat prostorové zvuky střevní peristaltiky v oblasti, kde se ve skutečnosti břicho nachází.

S parametrem zápachu je však situace oproti vizualizaci hladu o poznání komplikovanější. V realitě se u zápachu řídíme hlavně čichem, ten však ve VR v současnosti není možné simulovat. Haptika a sluch nám v tomto případě nikterak nepomohou. Zbývá tedy jen zrak. I když zápach není něco, co by se dalo lidským okem pozorovat, tak naštěstí i zde se lze inspirovat znázorněním, které je člověku dobře známé. Není zde nutné čerpat z reality, ale je možné vycházet z audiovizuálních děl. Zejména v kreslených filmech a seriálech se běžně stává, že tvůrci potřebují předat divákovi informaci o tom, že něco zapáchá. Tento problém se řeší za pomoci jakéhosi zeleného dýmu či oblaku, který z inkriminovaného místa vychází. Tímto způsobem se tedy pro potřeby této práce lze inspirovat.

Posledním parametrem je zdraví. Pro vizualizaci špatného vizuálního stavu je použito vykašlávání krve. Jedná se o kombinaci dvou smyslů: sluch (zvuk vykašlávání) a zrak (obraz krve). Díky zapojení více smyslů bude tento efekt účinnější a hráč se rychle dozví, že jeho virtuální charakter na tom není dobře.

3.2.3 Inventář

Inventář je důležitou součástí naší herní předlohy. Primárně se používá k úschově herních předmětů a jeho kapacita je omezená. Na obrázku č. 5 lze vidět, že inventář je realizovaný pomocí 2D GUI. Skládá se z polí uspořádaných do mřížky, mezi kterými se lze posouvat vertikálním posuvníkem.

U designu inventáře v rámci naší hry se stejně jako v předchozí sekci vyhneme pro VR nepraktickému GUI a znovu se inspirujeme ve skutečném světě, kde jsou pro dočasnou úschovu předmětů využívány batohy, tašky či různé kapsy od kalhot. V této práci je při realizaci použita pouze jedna z těchto možností – hráč si bude ukládat předměty pouze do virtuálních kapes.

Obrázek 5: Vizualizace inventáře v Hobo: Tough Life

3.2.4 Vybírání popelnic

Mezi klíčové herní prvky se řadí vybírání popelnic. Ve hře je zpracovaná tato mechanika jako abstraktní minihra formou 2D obrazovky. Jak lze vidět na obrázku č. 6, tento prvek je založený na principu odhalování polí, kde za každým z nich se skrývají užitečné herní předměty, ale také předměty, které po odhalení navyšují zápach postavy. Jedná se o penalizační nástroj za používání této mechaniky.

V Hobo: Tough Life se nachází vícero druhů popelnic s odlišnou velikostí a množstvím obsahu. I typy předmětů, které lze nalézt v popelnicích, jsou voleny podle konkrétního druhu popelnice.

Obrázek 6: Mechanika vybírání popelnic v Hobo: Tough Life

Mechanika v této práci je místo abstraktního pojetí upravena podle skutečných principů vybírání popelnic v prostoru. Místo odhalování polí se hráč bude muset skutečně přebírat v popelnici a hledat v ní předměty, které jsou pro něj užitečné. Navyšování hodnoty zápachu skrze předměty je nahrazeno penalizací za čas strávený při prohledávání dané popelnice.

3.2.5 Ohřívání u barelu

Mechanika ohřívání u barelu je tvořena kombinací 2D obrazovky a 3D prostoru kolem sudu. GUI se v tomto případě používá pro zapálení a udržování ohně přidáváním herních předmětů. Barel a prostor v jeho okolí (viz obrázek č. 7) slouží ke snižování úrovně promrznutí postavy a působí pouze, když je hráč v dostatečné blízkosti hořícího sudu.

Náhrazením udržování ohně pomocí 2D obrazovky skutečným přidáváním předmětů ve 3D, se zbylé fungování tohoto herního prvku může přenést i do VR a není třeba vymýšlet zcela odlišný design této mechaniky.

Obrázek 7: Ukázka ohřívání u barelu

3.2.6 Interakce s lidmi

Interakce s lidmi je v Hobo: Tough Life jedním ze základních stavebních kamenů hratelnosti. S lidmi se dá ve hře interagovat skrze dialogy. V pozadí každé komunikace je rozsáhlý větvený konverzační strom. Cílem interakce je získat herní předměty a peníze. Čím déle se hráči daří konverzovat, tím víc se zvedá jeho šance na úspěch a zlepšují se i odměny, jež lze případně získat. Při dialogu má hráč vždy na výběr z několika možností, kterými následně určuje směřování konverzace. Dialog je realizovaný použitím 2D GUI, jak lze vidět na obrázku č. 8.

Tento způsob interakce s lidmi si však žádá zobrazování velkého množství textů. Jedno z řešení, která se nabízí, je simplifikace konverzačních možností do podoby jednoduchých gest nebo symbolů. Ačkoliv takto můžeme vyřešit odpovědi volené hráčem, neřeší toto znázornění odpovědi od proti strany, s kterou dialog vedeme. Alternativní cestou celé mechaniky je vzít

původní cíl (snaha o získání předmětů od kolemjdoucích), změnit samotnou cestu k jeho dosažení a odstranit závislost této mechaniky na konverzování. Místo dialogů je tak možné použít gesto napodobující prosbu. V HTC Vive je obsažen mikrofon, a proto tuto mechaniku lze dále doplnit o zvukové prosby vycházející přímo od hráče.

Obrázek 8: Interakce s lidmi v Hobo: Tough Life

3.2.7 Nakupování

Ve hře lze nakupovat herní předměty v obchodech. Všechny obchody ve hře jsou realizovány pomocí 2D obrazovek. Oproti GUI prvkům v předchozích sekcích jsou obchody zobrazeny přes celou herní obrazovku, jak je patrné z obrázku č. 9. Jako alternativní design je pro obchody použito nahrazení 2D obrazovky skutečným obchodem ve 3D prostoru. Pro proces nákupu lze znovu využít hlasových příkazů, tentokrát však vycházejících z frází pro nakupování.

Obrázek 9: Ukázka nakupování v Hobo: Tough Life

3.2.8 Počasí

Proměnlivé počasí je v Hobo: Tough Life běžný jev a výrazně se podepisuje na hratelnosti. V herní předloze se nachází široké spektrum počasí (od slabých přeháněk až po silné bouřky či silný vítr, viz obrázek č. 10). Počasí negativně ovlivňuje parametry postavy. Pokud je hráč např. vystaven dešti, dochází k rychlejšímu prochladnutí. Samotný design systému počasí se nikterak nekříží s možnostmi VR a není zde třeba design jakkoliv upravovat. Pro zjednodušení se nebude v rámci této práce implementovat sněžení, ale pouze déšť.

Obrázek 10: Počasí v Hobo: Tough Life

4 Vývoj

Kapitola se zabývá technickým zpracováním, postupy použitými při vývoji a realizací designových návrhů z předchozí kapitoly.

4.1 Unity3D

Na vývoj této práce byl použit herní engine Unity ve verzi 2017.2. Ačkoliv existují i novější verze (2017.3 a 2018.1 v beta verzi) obsahující řadu vylepšení oproti 2017.2, byla dána přednost stabilitě a odladění starší verze před novými vlastnostmi a vylepšeními.

Jedná se o nejčastěji používaný program na tvorbu her pro augmentovanou a virtuální realitu. Je k dispozici zcela zdarma (i pro komerční účely), podporuje velkou škálu zařízení a platforem (např. Oculus Rift, Google Carboard, SteamVR, Xbox One, Playstation atd.), umožňuje tvorbu od malých mobilních her až po velkorozpočtové tituly, je přizpůsbený pro rychlé prototypování, klade důraz na rozšiřitelnost (velké množství doplňků, grafiky, pluginů atd., ke stažení přímo z vlastního online obchodu - Unity Asset Store) a je značně oblíben mezi vývojáři [21, 22].

Tato práce se podle zadání zaměřuje na platformu SteamVR, tudíž je pro práci s Unity využit SteamVR Plugin obsahující SteamVR Software development kit (SDK) vytvořený společností Valve Corporation na míru potřebám a specifikacím tohoto enginu.

4.2 Grafické zpracování

Tato sekce se zabývá obecnou strukturou grafiky celého projektu. Výsledný vzhled lze vidět na obrázku č. 11. Grafické detaily tykající se umělé inteligence, herních mechanik atd. jsou detailně popsány v podkapitolách. Pro potřeby této práce jsou použity nízko polygonální grafické modely od Synty Studio[1], které splňují požadavky stanovené v sekci 3.2.1.

V Unity existují dva standardní vykreslovací řetězce: takzvané opožděné vykreslování a dopředné vykreslování. Opožděné vykreslování nejprve vykreslí celou scénu bez světel a až posléze probíhá její nasvícení. Díky tomuto způsobu je možné navyšovat počet světel při vykreslování, aniž by se razantně zvedaly nároky na výkon. U dopředného vykreslování probíhá zpracování nasvícení již při samotném výpočtu jednotlivých vertexů a fragmentů. Pro každé světlo se nasvícení počítá zvlášť, navyšování světel tak značně zvedá nároky na výkon. Na druhou stranu na rozdíl od opožděného vykreslování umožňuje použití průhledností a při nízkém počtu světel je navíc rychlejší na zpracovaní než opožděné vykreslování [23].

Ve VR se obraz vykresluje stereoskopicky. V Unity se nacházejí tři metody pro toto zpracování: Multi Pass Stereo Rendering (MPSR), Single Pass Stereo Rendering (SPSR) a Single Pass Instanced Stereo Rendering (SPISR). MPSR je standartní vykreslování, kdy se obraz pro VR brýle zpracovává pomocí dvou samostatných kamer (reprezentující levé a pravé oko). Nejprve se vykreslí vše, co vidí první kamera, a následně se stejný proces opakuje pro druhou kameru.

[1] https://assetstore.unity.com/publishers/5217

Ačkoliv se jedná o implementačně nejjednodušší způsob, je značně neefektivní. Obě kamery jsou navzájem umístěny v těsné blízkosti a značná část scény se oboustranně překrývá. Pro většinu renderovaných objektů se tak zdvojnásobuje čas zpracování. Při SPSR se renderuje obraz pro obě oči najednou do společné vykreslovací textury. Scéna se tak zpracovává pouze jednou a zatížení CPU je výrazně sníženo. SPISR je pouze nadstavba SPSR, která přidává podporu instancování [24]. Současné verze Unity nemají plně dořešenou podporu SPSR (respektive SPISR) v odloženém vykreslování, kde při použití vznikají grafické artefakty a další problémy. Za účelem nejlepšího výkonu tedy byla pro tuto práci vybrána kombinace dopředného vykreslování se SPISR.

4.2.1 Optimalizace doby vykreslování

Pro současné VR zařízení je třeba snížit dobu strávenou vykreslováním výsledného snímku (který je následně zobrazen ve VR brýlích) pod 11 milisekund pro dosažení potřebných 90 FPS (viz 2.2.1). V Unity existuje celá řada optimalizačních technik určených pro zkrácení času, jež vykreslování zabere. Jedná se např. o static batching, occlusion culling, instancování, podporu LOD a mnoho dalších[25].

Static batching snižuje počet vykreslovacích instrukcí nad geometrií ve scéně pro objekty se stejným materiálem (entita obsahující shader a nastavení pro něj). Tato metoda se používá pro statické objekty, se kterými se za běhu aplikace nebude pohybovat (např. budovy). Interně funguje static batching na základě transformování souřadnic statických modelů do světových souřadnic, z nichž se vytváří jeden společný vertex a index buffer, který se následně používá při vykreslování místo původní geometrie [26].

Occlusion Culling je technika deaktivující renderování objektů, které jsou z pohledu kamery neviditelné, jelikož jsou blokovány jiným objektem. Výhodou této metody je eliminování problému překreslování (anglicky overdraw), ke kterému dochází postupným vykreslováním grafiky od nejvzdálenější geometrie po nejbližší, kdy se výpočetní výkon zbytečně plýtvá na věci, které nejsou ve výsledném obrazu vidět [27].

Instancování na grafické kartě se používá k vyrenderování stejných objektů jako jeden vykreslovací příkaz. Na rozdíl od static batchingu je tato technika aplikovatelná na dynamické objekty ve scéně (kromě objektů využívajících takzvaný vertex skinning) [28]. Mechanika se tak využívá na objekty, které vznikají až za chodu samotné hry (např. herní předměty).

Ačkoliv v Unity existuje podpora pro LOD (označení systému, který s přibývající vzdáleností objektu od kamery nahrazuje topologii tělesa simplifikovanými objekty obsahujícími méně polygonů), na rozdíl od herních programů jako Unreal Engine či CryEngine zde nelze jednotlivé LOD tvořit automaticky, a k jejich vytvoření tak musíme použít jiné nástroje než ty zabudované [29]. Pro potřeby této práce byla použita kombinace ruční tvorby LOD v Blenderu[2] a poloau-

[2]https://www.blender.org

tomatické zpracování pomocí nástroje Mantis LOD[3]. Tento systém není aplikovaný pro každý prvek ve scéně, ale jen pro objekty, jejichž topologie přesahuje 10 tisíc vrcholů.

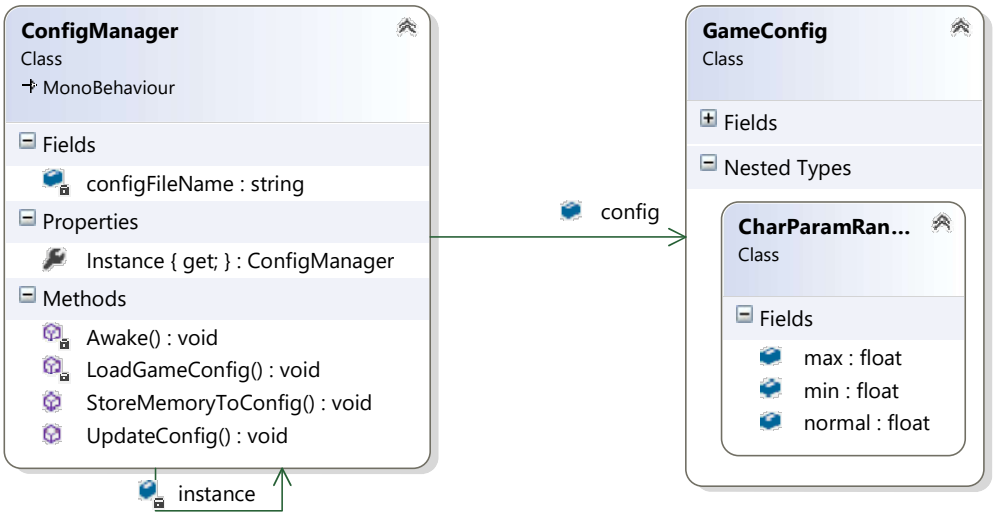
Obrázek 11: Ukázka grafiky městského prostředí z výsledné aplikace

4.3 Nastavení z konfiguračního souboru

Veškeré nastavení herních mechanik se načítá z konfiguračního souboru s názvem `config.json`. Struktura tohoto souboru je definována třídou `GameConfig`. Pro převedení této struktury na disk se provádí serializace do formátu JavaScript Object Notation (JSON) pomocí třídy `ConfigManager`. Tato třída při spuštění hry deserializuje nastavení ze souboru do instance třídy `GameConfig`. Jak lze vidět na obrázku č. 12, reference na tuto instanci je v `ConfigManager` uchována, díky čemuž lze k jednotlivým parametrům v nastavení přistupovat globálně.

Pro usnadnění práce při ladění a nastavování hodnot v configu byly přidány funkce umožňující regenerování souboru a možnost přepsat obsah dat na disku aktuálním obsahem v paměti. Tyto metody jsou napojené na API Unity Editoru a lze je tak používat přímo v prostředí enginu.

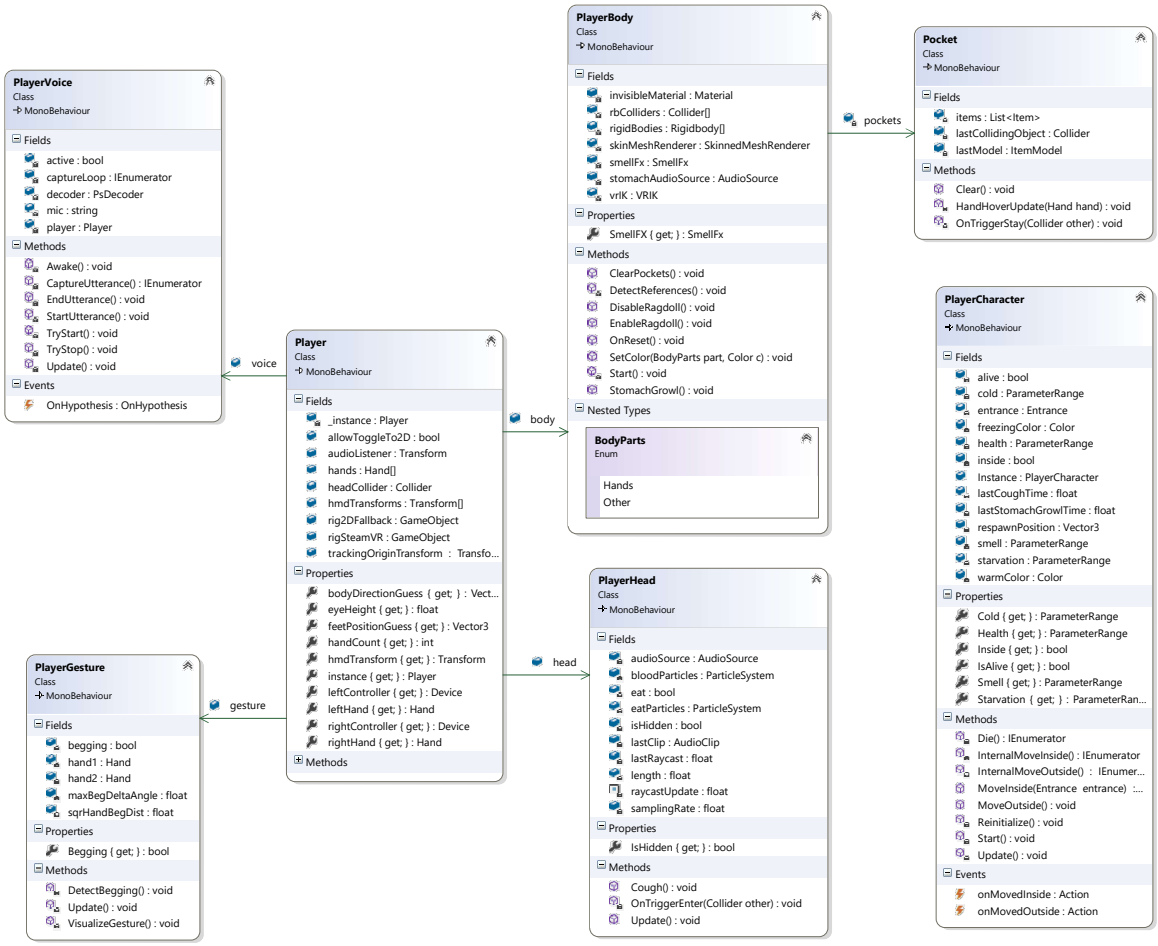
[3] http://www.mesh-online.net/index.html



Obrázek 12: Třídní diagram konfigurace ze souboru

4.4 Herní postava

Herní postava se skládá z několika dílčích komponent: RPG logika, pohyb v prostoru, tělo postavy, hlava postavy, gesta a rozpoznávání řeči. Tyto komponenty vycházejí z designu a návrhu v kapitole 3. Až na RPG logiku se všechny prvky vztahují na fyzickou reprezentaci hráče ve VR. Ve snaze o přehlednou a snadno dostupnou strukturu byla pro třídy reprezentující tyto komponenty vytvořena vazba na třídu `Player`, která je již obsažena ve SteamVR Pluginu pro Unity a slouží jako prostředník pro přístup k VR brýlím, ovladačům a hernímu prostoru kolem uživatele. Všechny prvky související s reálným hráčem a jeho 3D replikou ve VR jsou tak dostupné z jednoho místa a RPG logika zůstává stranou. Strukturu jednotlivých komponent a vazeb mezi nimi lze vidět na obrázku č. 13.



Obrázek 13: Třídní diagram hráče

4.4.1 RPG logika

Pro logiku RPG prvků herního charakteru byla vytvořena třída `PlayerCharacter`. V této třídě jsou obsaženy veškeré herní parametry definovány v kapitole 3.2.2. Nacházejí se zde tedy parametry reprezentující život, zápach, hlad a úroveň prochladnutí, které jsou realizovány formou instančních proměnných typu `ParameterRange` (třída představující interval s definovanou minimální a maximální hodnotou a proměnnými reprezentujícími momentální hodnotu a hranici, kdy se interval nachází v normě). Nastavení pro tyto jednotlivé veličiny se odebírá z konfiguračního souboru v třídě `ConfigManager`. V rámci třídy `PlayerCharacter` se hodnoty jednotlivých veličin průběžně mění. Úroveň prochladnutí a hladu v čase roste konstantní rychlostí, která je pro každý parametr zvlášť definovaná v konfiguračním souboru. Na navyšování hodnoty prochladnutí se vyjma průběžného růstu projevuje rovněž momentální stav počasí, přičemž přírůstek se vypočítává z aktuálního množství srážek × konstantní síla deště × celočíselný přepis proměnné typu bool, která reprezentuje, zda je hráč vystaven dešti, či nikoliv. Nad jednotlivými parametry

(respektive instancemi třídy `ParameterRange`) se pomocí funkcí `AboveNormal` a `BelowNormal` průběžně provádí kontrola, jejíž cílem je zjistit, zda současná hodnota parametrů v podobě interní proměnné `value` odpovídá v nich stanovené normě. V situaci, kdy je tato hranice současnou hodnotou překročena, se pro tuto entitu vypočítává její momentální normalizovaná hodnota představující reálné číslo v intervalu od 0 do 1, kde 0 odpovídá minimální hranici pro překročení normy a 1 se rovná maximální hodnotě intervalu pro daný objekt. Tato normalizovaná hodnota se používá jako parametr v interpolaci při grafické vizualizaci prochladnutí a zápachu mezi jejich výchozím stavem (např. přirozená barva kůže) a výsledným vizuálním znázorněním při dosažení maximální hodnoty těchto parametrů (zbarvení pokožky do modra atd.). Dále se tato hodnota používá při výpočtu poklesu zdraví, který se rovná součtu všech takto normalizovaných parametrů × konstantní síla úbytku zdraví.

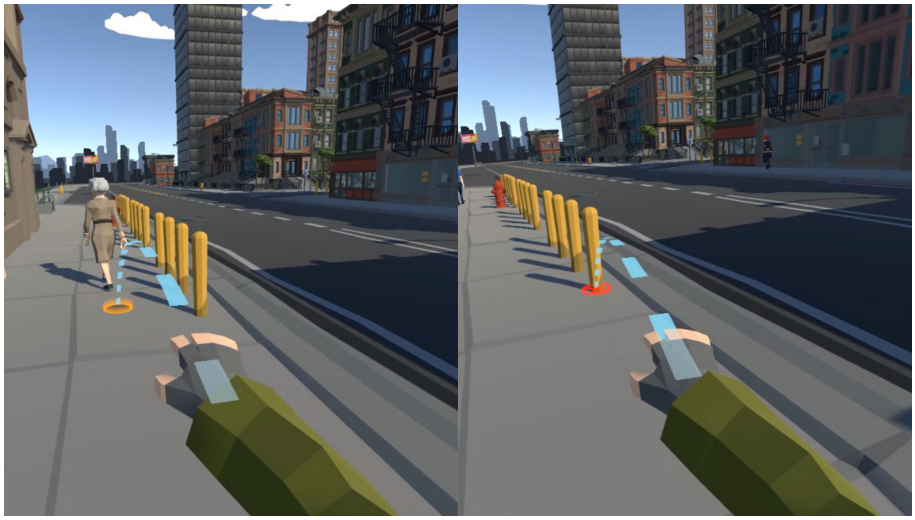
4.4.2 Pohyb v prostoru

Zařízení HTC Vive a Oculus Rift umožňují prostorové snímání lokomoce VR brýlí a ovladačů. Lze tak zachytit pozici a rotaci hlavy hráče a jeho rukou. Prostor, ve kterém lze tyto lokomoce zachycovat, je však omezený. Tento problém je dán současným technickým omezením senzorů, které v případě HTC Vive umožňují sledování pohybu v oblasti o maximální velikosti $4,5m^2$ a v případě Oculus Rift jen $2,5m^2$, přičemž rozsah snímání je dále limitován samotným prostorem, v němž uživatel tato VR zařízení používá [30]. Pro realizaci volného pohybu v rámci našeho herního světa, který svou velikostí překračuje možnosti hardwarového snímaní, se v rámci této práce experimentovalo se dvěma metodami umožňujícími pohyb ve větším prostoru, než jaký nabízí dané zařízení. První zkoumanou metodou byla technika vycházející z fyzického pohybu, kdy se snažíme na místě simulovat pohyb dopředu ve VR. Druhou zvažovanou metodou byla takzvaná teleportace, kdy se hráč pohybuje v rámci vymezeného prostoru a pro přesun mimo tuto vymezenou oblast využívá přemístění celé zóny pohybu na jím vybrané místo. V obou případech se jedná o běžně používané techniky ve VR hrách. Problémem prvního přístupu je, že pro správné fungování pohybu na místě je vždy zapotřebí provést kalibraci výšky uživatele a nastavení citlivosti detekce lokomoce skrz pohyb ovladačů a VR brýlí. Navíc se s tímto řešením pojí úskalí se setrvačností pohybu. Jedná se například o situaci, kdy se rychlým pohybem na místě snažíme simulovat běh rovně, který se následně rozhodneme zastavit. Na naše tělo sice v této chvíli nepůsobí setrvačnost z pohybu, jelikož jsme se po celou dobu nacházeli na jednom místě, a zastavení je tak téměř okamžité, ale kvůli zpoždění, jež vzniká při replikaci pohybu hráče na základě postupného vyhodnocování změn pozic a rotací ovladačů a VR brýlí, se setrvačnost může projevit ve VR. Dochází tak ke kolizi ve vnímání pohybu, která může vést k vyvolání kinetózy [31]. Z těchto důvodů byla upřednostněna teleportace.

Pro potřeby této práce se na teleportaci používá ViveTeleporter[4]. Jedná se o teleportační systém vytvořený pro použití v enginu Unity na platformu SteamVR. Pro určení místa teleportace

[4]https://github.com/Flafla2/Vive-Teleporter

se využívá touchpad na ovladačích, který při zmáčknutí aktivuje prostorový ukazatel obloukovitého tvaru. Tento ukazatel ve směru natočení ovladače provádí za pomoci fyzikálního systému v enginu ověřování zda (a případně kde konkrétně) dochází ke kolizi s fyzikální reprezentací herního světa. Pokud je tímto způsobem nalezen kolizní bod, tak se na jeho pozici zobrazí náš teleportační ukazatel a v rámci jeho blízkého okolí se provádí hledání v takzvaném navigačním meshi. Ten reprezentuje zjednodušenou reprezentaci dostupného prostoru, který Unity generuje na základě geometrie herního světa [32]. Pokud se v daném místě navigační mesh nenachází, je vizualizace ukazatele změněna na červený kříž představující nedostupnou oblast pro teleportaci (viz obrázek č. 14).



Obrázek 14: Ukázka teleportace

4.4.3 Tělo postavy

Tělo charakteru se skládá ze tříd `VRIK`, `NavMeshObstacle`, `PlayerBody` a 3D modelu herní postavy. `VRIK` je třída obsažená v Unity pluginu s názvem FinalIK[5]. Jedná se o rozšíření sloužící pro transformaci kloubů ve skeletonu modelu tak, aby orientace těchto jednotlivých spojů byla uspořádaná do tvaru, kdy se poslední článek shoduje s námi požadovanou transformací (například úprava pozic kloubů ruky v modelu do tvaru pravého úhlu s cílem uchopení předmětu ve výšce pasu postavy). Tento proces se nazývá inverzní kinematika [33]. Konkrétně `VRIK` slouží na aproximaci pozice a rotace všech kostí a kloubů humanoidního modelu na základě aktuální transformace VR brýlí a ovladačů. Za pomoci této entity je možné promítnout reálné pohyby rukou, nohou a hlavy na hráčský model postavy. `NavMeshObstacle` je třída obsažená v Unity API. Její využití je popsáno v kapitole 4.5.3. Třída `PlayerBody` obsahuje vazby na `VRIK`, kolizní zóny těla, inventář a efekt zápachu. Skrz instanci `PlayerBody` se tyto jednotlivé reference spravují.

[5] https://assetstore.unity.com/packages/tools/animation/final-ik-14290

Model od Synty Studio vybraný pro postavu hráče bylo nutné s pomocí blenderu rozdělit do tří částí (hlava, ruce a zbylé časti těla), aby bylo možné pro každý z těchto dílů využít jiný materiál. Pro oblast hlavy byl použit materiál s shaderem, který do výsledného obrazu vykresluje pouze stín, ne však samotný model (viz obrázek č. 15). V těsné blízkosti hlavy je totiž umístěna pohledová kamera, a docházelo by tak k prolínání 3D modelu a obrazu zachyceného touto kamerou. Ruce byly od těla odděleny kvůli vizualizaci prochladnutí. Potřebujeme mít totiž možnost dynamicky měnit barvu modelu z původní barvy na modrou. Pokud by tyto dvě části zůstaly pohromadě, změna barvy by se projevovala i na oblečení. Rozdělením tak můžeme upravovat barvu přímo na pokožce. Objektová reprezentace modelu ve scéně enginu Unity obsahuje v rámci své skeletální struktury kolizní zóny (viz obrázek č. 15). Ty jsou při hraní deaktivovány do doby, než hráčovo zdraví klesne na hodnotu 0, kdy nastává úmrtí virtuální postavy. Při aktivaci jednotlivých zón dochází k deaktivaci VRIK, a model je tak ovlivňován výhradně fyzikou. Herní postava pod tíhou gravitace padá k zemi a tělo se deformuje na základě kolize s okolním světem.

Obrázek 15: Tělo postavy v Unity Editoru

4.4.3.1 Třída GripAnimator Ve hře se hráč často ocitá v situaci, kdy potřebuje něco uchopit (například herní předmět). Plugin FinalIK má na starost transformaci končetin, avšak stisknutí pěstí nepodporuje. Pro tento případ tak byla vytvořena třída `GripAnimator`. Pro každý ovladač (představující hráčovu ruku) existuje jedna instance této třídy. Každá z nich obsahuje pole rotačních kvaternionů jednotlivých článků prstů na ruce jak ve výchozím stavu, tak ve stavu při úplném sevření dlaně. Mezi těmito dvěma stavy se provádí lineární interpolace, přičemž interpolačním parametrem je aktuální hodnota z osy x na triggeru ovladače, která nabývá reálných čísel od 0 do 1. Díky tomuto způsobu jsme tak schopni věrně replikovat úchop ruky ve VR, jak lze vidět na obrázku č. 16.

Obrázek 16: Průběh animace úchopu

4.4.3.2 Inventář Jak již bylo stanoveno v návrhu, inventář je realizovaný pomocí ukládání předmětů do kapes. Ty jsou ve hře reprezentovány třídou `Pocket`. Na těle postavy (konkrétně u pasu z levé a pravé strany) se nacházejí dvě instance této třídy. Ke každé z nich je přidružený kolizní objekt. Jakmile se v této kolizní oblasti nachází předmět, který není nikterak vázaný na jiný objekt (např. není držen v ruce atd.), dochází ke zničení jeho prostorové podoby a je přidán do interní kolekce herních předmětů v rámci entity třídy `Pocket`. Při dosažení maximální kapacity kapsy toto neplatí. Pro vyjmutí předmětu z kapsy je třeba, aby se ovladač, přesněji řečeno instance třídy `Hand` reprezentující ovladač, dostal do kolizní zóny. Pokud se tedy ovladač nachází v kolizní oblasti kapsy a dojde ke zmáčknutí triggeru, tak se z interní kolekce odebere naposledy vložený předmět, vytvoří se jeho 3D podoba a dojde k přiřazení pod instanci ruky, která akci vyvolala.

4.4.3.3 Vizualizace zápachu Na realizaci efektu zápachu dle návrhu (formou zeleného kouře) se využil takzvaný particle systém (interní systém v Unity sloužící pro vykreslení velkého počtu malých a jednoduchých obrazců formou částic, které dohromady tvoří amorfní či tekutý tvar [34]). Při vývoji této vizualizace se experimentovalo s několika různými tvary, barvami a efekty pro částice v tomto particle systému. Finální varianta se skládá z částic kruhovitého tvaru, které nabývají různých odstínů zelené a náhodné velikosti (viz obrázek č. 17). V čase se jednotlivé částice zmenšují a stoupají směrem vzhůru, aby připomínaly skutečný kouř. Na modelu postavy (konkrétně na rukou) jsou rozmístěné celkem čtyři takovéto particle systémy. Pro správu vizualizace aktuální úrovně zápachu byla vytvořena třída `SmellFx`. Množství vypouštěných částic ze všech particle systémů určuje instance této třídy přenásobením maximálního množství emisí pro daný systém normalizovanou hodnotou parametru zápachu.

Obrázek 17: Vizualizace efektu zápachu

4.4.4 Hlava postavy

Pro logiku týkající se oblasti hlavy postavy vznikla třída `PlayerHead`, jež se stará o konzumaci herních předmětů a efekt vykašlávání krve. Konzumace je realizovaná formou kolizní zóny v oblasti rtů. Jestliže do této zóny vstoupí předmět určený ke spotřebování (tzn. jídlo, pití nebo medikamenty), tak dochází k okamžité konzumaci (použití a odstranění předmětu viz kapitola 4.6 věnující se předmětům). Toto spotřebování je v případě pití podpořeno zvuky srkání (které byly vlastnoručně vyrobeny). Pakliže se jedná o jídlo, přehrávají se zvukové efekty mlaskání získané z databáze AudioBlocks[6]. V závislosti na délce přehrávaného zvuku jezení se pak pomocí particle systému generují částice v oblastí rtů simulující drobky vzniklé při konzumaci. Na efekt vykašlávání krve byl opět použit particle system, který je navíc rozšířen o kolizi a takzvané sub-emittory (částice vzniklé při kolizi původních části s jiným kolizním objektem). Z oblasti rtů tak při vykašlávání vycházejí fluidní částice zbarvené do červena, které při kolizi s okolním herním světem vytvářejí efekt rozprsknutí. Částice jsou emitovány ve vlnách, které jsou synchronizovány se zvukem kašlání[7].

4.4.5 Gesta

Na detekci hráčových gest byla vytvořena třída `PlayerGesture`. V rámci této práce se používá gesto prošení (viz obrázek č. 18) pro mechaniku žebrání. Detekce tohoto gesta funguje na principu porovnávání definovaných instančních proměnných vzhledem k aktuální rotaci ovladačů a jejich vzájemné vzdálenosti. Jakmile se vzdálenost mezi nimi a jejich natočením přiblíží cílovým hodnotám v proměnných na minimální úroveň tolerance, dochází k pozitivnímu vyhodnocení gesta pro žebrání.

[6] https://www.audioblocks.com/stock-audio/cartoon-ungly-eating.html
[7] https://www.zapsplat.com/sound-effect-category/cough/

Obrázek 18: Ukázka gesta žebrání

4.4.6 Rozpoznávání řeči

Pro hlasové pokyny u mechanik žebrání či nakupování v obchodech byla použita open source knihovna na rozpoznávání řeči CMUSphinx[8], respektive její odlehčená verze UnitySphinx[9] upravená pro použití v rámci enginu Unity. Tato verze obsahuje hlasový dekodér, který provádí analýzu zvukové stopy na základě akustického modelu a slovníku výslovnosti. Pro CMUSphinx existuje v současnosti 12 oficiálních jazykových sad[10]. Jedná se například o němčinu, italštinu nebo ruštinu. Pro svou globální rozšířenost byla pro tuto práci vybrána jazyková sada americké angličtiny. Za účelem zkombinování zvukového vstupu, jeho analýzy a zpracování byla vytvořena třída `PlayerVoice` vycházející z ukázkových tříd obsažených v UnitySphinx. Ve hře existuje pouze jedna entita této třídy. Při její inicializaci dochází k vytvoření instance dekodéru a provedení jeho konfigurace (například napojení na akustický model atd.). Při zmáčknutí tlačítka grip na kterémkoliv z ovladačů, dochází k zahájení nahrávání z mikrofonu ve VR brýlích do zvukové stopy v paměti. Po dobu, kdy je toto tlačítko drženo, probíhá dekódovací analýza, která porovnává záznam s jednotlivými příkazy a větnými celky obsaženými v souboru `commands.gram`. Průběžné nahrávání z mikrofonu je zastaveno, jakmile dojde k uvolnění tlačítka grip. Po ukončení nahrávání dochází k finální analýze, jejímž výsledkem je hypotézový převod zaznamenaného zvuku do textového formátu typu string a číselné ohodnocení jistoty dekodéru touto analýzou. Tento textový přepis a ohodnocení jsou použity jako parametry v následně vyvolané veřejně dostupné instanční události.

[8] https://cmusphinx.github.io
[9] https://github.com/pdehn/UnitySphinx
[10] https://sourceforge.net/projects/cmusphinx/files/Acoustic20and20Language20Models

4.5 Umělá inteligence

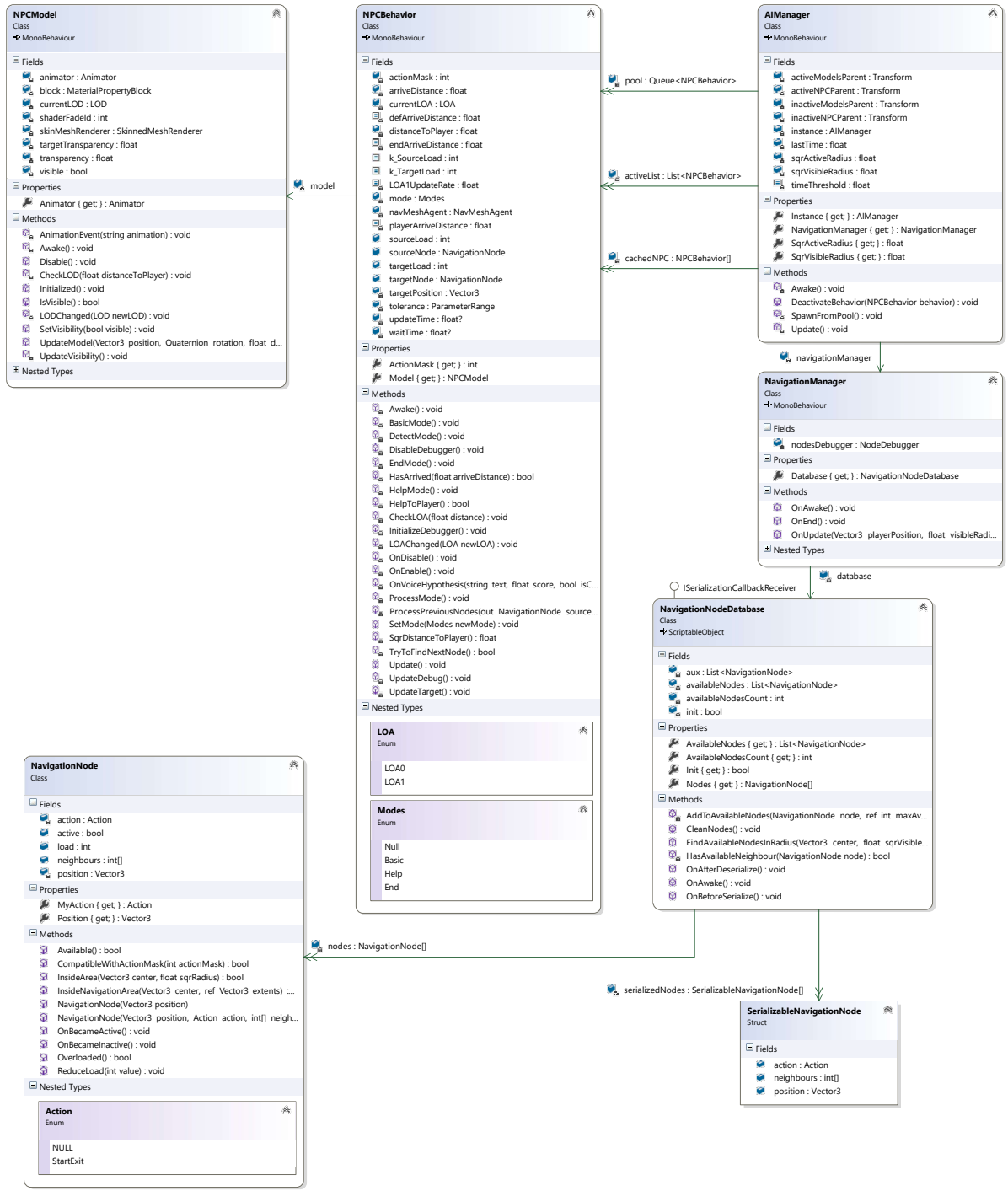
Tato podkapitola se věnuje umělé inteligenci (anglicky Artificial Intelligence, zkratka AI) ve hře. V rámci této práce se tímto termínem myslí takzvané Non-Player Characters (označení pro postavy, které se vyskytují v herním světě, ale nejsou ovládané hráčem a mají své vlastní chování, zkratka NPC). Jedná se o lidi chodící po ulicích města. Jednak slouží pro dotvoření atmosféry opravdového městského prostředí a za další se využívají jako herní mechanika pro získání herních předmětů prostřednictvím žebrání. Jednotlivé třídy tvořící AI a vazby mezi nimi lze vidět na obrázku číslo 19. Na tomto obrázku se nacházejí pouze třídy, které se používají za běhu hry. Při vývoji však bylo nutné navíc naprogramovat pomocné nástroje pro snadnou manipulaci s těmito třídami v rámci prostředí enginu Unity. Ty jsou společně s ostatními třídami popsány v následujících sekcích.

4.5.1 Řídící manažer umělé inteligence

Pro správu a řízení umělé inteligence vznikla třída `AIManager`. Jejím cílem je, aby se jednotlivá NPC nacházela v blízkém okolí hráče. Bylo by totiž zbytečným plýtváním využívat výpočetní prostředky na entity, které jsou mimo dosah hráče (například na opačné straně herního světa). Primárním úkolem instance této třídy je tedy vytváření a zánik jednotlivých NPC. Oblast, ve které se v danou chvíli mohou NPC vyskytovat, je definována pomocí dvou kruhů se středem na pozici hráče. První kruh označuje takzvanou viditelnou zónu. V rámci ní jsou jednotlivá NPC pro hráče viditelná a hráč s nimi může interagovat. Druhý kruh vyznačuje aktivní zónu. Ta má poloměr větší než viditelná oblast. Uvnitř této zóny jsou NPC stále ještě aktivní. Nejsou však pro hráče viditelná a nemusí se tedy vykreslovat. Po teleportaci se díky dvou zónovému rozdělení v okolí hráče stále nacházejí NPC. Hráč se totiž přesunul do míst, ve které NPC byly neviditelné, avšak stále aktivní. Není tak třeba využívat CPU a GPU na větší vykreslovací vzdálenost jednotlivých NPC. Pro editaci a vizualizaci těchto zón při vývoji byla vytvořena třída `AIManagerEditor`, která za pomocí API Unity Editoru vykresluje jednotlivé hranice oblastí a umožňuje dynamicky měnit jejich poloměr uvnitř Unity Editoru v rámci herní scény.

Vytváření NPC za běhu je časově náročnou operací (instancování NPC, inicializace v nich obsažených entit, načtení a instancování 3D modelu z disku atd.), při níž by docházelo k propadům FPS. Pro vznik a zánik NPC byl z tohoto důvodu použit návrhový vzor Object pool. Tento vzor představuje koncept takzvaného poolu. Jedná se o entitu obsahující interní kolekci jiných objektů. Instance mající přístup k tomuto poolu si tak místo náročného instancování mohou vyžádat již dříve vytvořený a inicializovaný objekt, který je obsažený v rámci poolu [35]. Spojení tohoto vzoru se správou NPC je realizováno v rámci třídy `AIManager` pomocí kombinace tří kolekcí. Všechny tyto kolekce jsou určeny pro objekty typu `NPCBehavior` (třída reprezentující jednotlivá NPC, viz podkapitola 4.5.3). Konkrétně se jedná o instanční proměnné `cachedNPC` (Array), `activeList` (List) a `pool` (Queue). V rámci prostředí Unity editoru jsou jednotlivá NPC předvytvořeny a za pomoci serializace přidána do pole `cachedNPC` v instanci

třídy `AIManager`. Při spuštění samotné hry se obsah tohoto pole převádí do instanční proměnné pool. Při požadavku na vytvoření se z této kolekce odebere NPC, které je přidáno do kolekce v proměnné `activeList`. Ta představuje momentálně aktivní NPC. Jakmile přestává být NPC aktivní, je přidáno zpět do poolu.



Obrázek 19: Třídní diagram umělé inteligence

4.5.2 Navigační síť

Aby vytváření NPC v okolí hráče a jejich následný pohyb v prostoru nebyl zcela chaotický a náhodný, byla vytvořena navigační síť, která je realizovaná třídou `NavigationNodeDatabase`. Tato síť je tvořena z navigačních uzlů (instancí třídy `NavigationNode`). V rámci entity třídy `NavigationNodeDatabase` jsou pak tyto uzly obsažené v instanční proměnné typu pole pod názvem `nodes`. Každý uzel má definovanou pozici, akci (například že se jedná o ukončovací uzel), pole indexů sousedících uzlů, aktuální zatížení (dáno množstvím entit, které uzel využívají) a zda je, či není momentálně v aktivní oblasti. `NavigationNodeDatabase` dědí ze třídy `ScriptableObject`. Jedná se o třídu v UnityEngine API umožňující serializovat do souboru velké množství sdílených dat. Hlavní výhodou je redukování využití paměti zamezením tvorby kopií jednotlivých hodnot. Další výhodou je jednoduché tvoření vazeb v prostředí enginu Unity [36].

Pro editaci sítě uvnitř editoru (např. tvorba nových uzlů, tvoření vazeb mezi nimi, jejich přemísťování atd.) byla vytvořena třída `NavigationNodeDatabaseEditor`. Ta v prostředí Unity Editoru, při vybrání souboru obsahujícího serializovaná data instance NavigationNodeDatabase, v rámci scény a inspectoru (panelové okno v Unity Editoru sloužící pro grafickou vizualizaci objektů, scriptů, modelů, textur atd.) zobrazí editační nástroje a vizualizaci sítě (viz obrázek č. 20). Uvnitř scény se tak vykreslují jednotlivé uzly, vazby mezi nimi a aktuální index v poli. Podle typu akce uzlu se navíc na jeho pozici vykreslí ikonka specifická pro danou akci. Jednotlivé uzly lze vybírat najetím kurzoru do oblasti v okolí jejich pozice. Vybraný uzel je napojený na entitu třídy `Handels` (interní třída enginu sloužící pro posuvníky obsažené v API Editoru), a lze jej tak posouvat pomocí posuvníků. Přidání a odebírání vazeb je realizováno v takzvaném editačním módu. Před jeho použitím je třeba mít vybraný uzel. Pro aktivaci (případně deaktivaci) editace je třeba zmáčknout klávesu E. V rámci editace se po kliknutí na libovolný uzel ve scéně přidá (či případně zruší) vazba mezi kliknutým uzlem a tím který byl původně vybraný. U obou tedy dojde k rozšíření pole sousedů o index toho druhého. Kromě obousměrných vazeb lze zvolit i jednosměrné, kdy se pole sousedů rozšíří o index pouze u vybraného uzlu. Pro usnadnění práce v rámci scény byla přidána možnost vycentrovat kameru na aktuálně vybraný uzel pomocí klávesy F. Tato vlastnost je realizovaná použitím funkce `LookAt()` z Unity API, které kameru transformuje na požadovanou pozici.

Aktuálně vybraný uzel a jeho sousední uzly se taktéž zobrazují v inspectoru. Pro každý uzel se zde nachází rozbalovací nabídka na změnu akce, editovatelný přepis pozice a tlačítka umožňující uzel smazat či jej zvolit jako aktuálně vybraný. Dále je zde tlačítko na přidávání nových uzlů. Při jeho zmáčknutí se vezme aktuální transformace kamery ve scéně a z jejího středu je veden paprsek (označení pro nekonečnou přímku, definovanou výchozím bodem a směrem) do scény, který hledá nejbližší kolizní bod s herním světem. Pokud v daném směru nastane kolize, umístí se na kolizní bod nový uzel. Jestliže k žádné kolizi nedošlo, uzel se vytvoří na výchozí pozici paprsku.

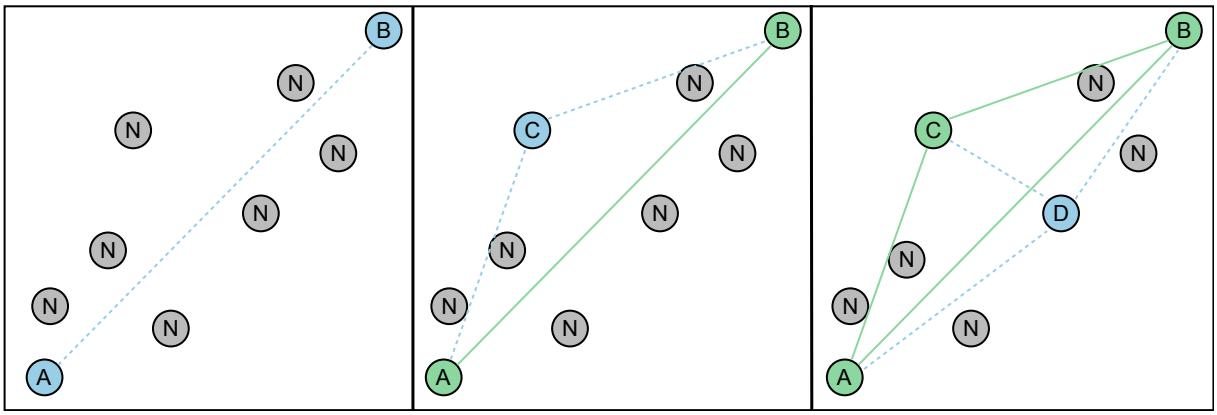
Obrázek 20: Ukázka editačního nástroje navigační sítě

Komunikace mezi `AIManager` a navigační sítí je zajištěna pomocí třídy `NavigationManager`. Ta v sobě obsahuje vazbu na instanci třídy `NavigationNodeDatabase`. Stará se o čištění uzlů (například vynulování zátěže atd.) za chodu aplikace a průběžně aktualizuje dostupné navigační uzly voláním funkce `FindAvailableNodesInRadius()` nad objektem navigační sítě. V rámci této funkce předává síti tyto parametry: aktuální pozice hráče a rádie viditelné a aktivní oblasti z instance `AIManager`. Objekt `NavigationNodeDatabase` na základě těchto parametrů prochází všechny své uzly a určuje, zda jsou aktivní a to na základě porovnání vzdálenosti pozice uzlu od hráče vzhledem k poloměru aktivní zóny. Jestliže je uzel aktivní, provádí se navíc kontrola, zdali je i dostupný. Dostupné uzly se totiž používají v rámci `AIManager` pro vytváření NPC. Ověřuje se tedy, zda není přetížený (momentální zatížení přesahuje definovanou hranici) a zda obsahuje vazbu alespoň na jeden uzel, který taktéž není přetížený. Pakliže se nejedná o zahajovací či ukončovací uzel (které se používají například u vchodu do budovy), provádí se ještě kontrola, zda uzel není obsažen v rámci viditelné zóny, aby například nedocházelo k vytváření NPC uprostřed ulice hned vedle hráče. Uzel, který splňuje všechny tyto podmínky, se přidá do kolekce `availableNodes`. Po průchodu všech uzlů je provedeno třídění této kolekce podle aktuální zátěže v ní obsažených objektů pomocí algoritmu MergeSort. Nejedná se o náročnou operaci, jelikož dostupných uzlů je vzhledem k velikosti aktivní oblasti a hustoty sítě vždy jen pár. Tuto seřazenou kolekci dostupných uzlů průběžně prochází `AIManager`, a pokud má k dispozici volné NPC uvnitř poolu, vytvoří na pozici tohoto uzlu NPC. Jelikož NPC po vytvoření tento uzel dále používá (viz podkapitola 4.5.3), je navýšeno zatížení uzlu, přičemž při dalším vytváření v rámci `AIManager` dostane přednost uzel s nižším zatížením.

Využívání zatížení samo osobě nezajišťuje vyrovnanou distribuci NPC v navigační síti. Instance třídy `AIManager` proto při vytváření prochází dostupné uzly sekvenčně. Pokud totiž v rámci sítě existují uzly, které jsou si blízké a v rámci kolekce je rozestup mezi jejich indexy

malý, pak by při vytváření NPC některé části sítě byly více využívány i přes rovnoměrné rozmístění samotných uzlů. Řešením není použití náhodného přístupu k objektům uvnitř kolekce `availableNodes`, jelikož se jedná o kolekci tříděnou podle zátěže v ní obsažených uzlů. Pří náhodném přístupu by tak mohly být upřednostněny uzly s vyšším zatížením. Možným řešením by například bylo nahrazení kolekce `availableNodes` polem polí. Hodnoty s indentickou zátěží by byly vloženy do společného pole. Nad tímto polem by pak bylo možné provádět náhodný přístup. Tato metoda však vyžaduje delší dobu na zpracování. Dalším problémem je samotný náhodný přístup. Ten totiž nemusí vždy zajistit optimální rozvrstvení zátěže. Vzdálenostně blízké uzly totiž často mají stejné zatížení a i při náhodném výběru by mohlo dojít k nevyváženému rozdělení. Byl tedy vytvořen algoritmus, jehož cílem je upravit řazení v rámci pole `nodes`, tak aby tato kolekce byla rovnoměrně zatížená (viz obrázek č. 21). Při hledání dostupných uzlů již budeme pracovat s vyváženým rozmístěním. Zpracovaná kolekce `availableNodes` tedy půjde projít sekvenčně, aniž by došlo k narušení rovnoměrnosti. Algoritmus postupuje následovně:

1. nalezení uzlu s nejmenší vzdáleností od minimálního bodu (3d vektor se složkami rovnající se nejmenší hodnotě floatu) a vložení do pole zpracovaných uzlů na pozici 0
2. nalezení uzlu s největší vzdáleností od uzlu z bodu č. 1 a vložení do pole na pozici o jednu větší
3. zpracování zbývajících uzlů
   (a) spočítání procentuálního podílu nejmenší vzdálenosti uzlu na součtu všech vzdáleností uzlu od již zpracovaných uzlů
   (b) vložení uzlu s největší procentuální hodnotou do zpracovaných uzlů
   (c) opakování bodu č. 3



Obrázek 21: Průběh postupného třídění uzlů. Šedá barva označuje nezařazené uzly. Písmena barevných uzlů označují umístění v kolekci zpracovaných uzlů

4.5.3 NPC

Logika NPC je zastřešená třídou `NPCBehavior`. Chování NPC je v rámci této třídy realizováno pomocí tří takzvaných módů chování. Základním z nich je mód s názvem Basic. Jeho cílem je dostat NPC ze zdrojového uzlu (který je pro jednotlivé instance nastavený při vytvoření) do cílového uzlu. Jelikož má NPC po vytvoření definovaný pouze zdrojový uzel, dochází k zavolání funkce `TryToFindNextNode()`. Jejím účelem je nalezení cílového uzlu mezi sousedícími uzly zdrojového uzlu. Při hledání se klade důraz na momentální zatížení jednotlivých uzlů. Po vybrání cílového uzlu je aktivován pohyb na jeho pozici. Samotný pohyb je realizovaný pomocí třídy `NavMeshAgent`, na kterou je objekt `NPCBehavior` navázaný. Jedná se o interní třídu v Unity API, která se stará o přesouvání se směrem k definovanému cíli a také o vyhýbání se ostatním instancím této třídy a instancím třídy `NavMeshObstacle` (instance, která zamezuje pohyb přes svou pozici, využita například u hráče). Pro pohyb třída `NavMeshAgent` využívá navigační mesh (viz podkapitola 4.4.2). Nad tímto meshem průběžně provádí hledání cesty k cílové pozici, po které se následně přemisťuje [37]. Jakmile se pozice NPC rovná cílové pozici, tak se v rámci Basic módu provádí zpracování cílového uzlu. Pokud nemá uzel žádnou konkrétní akci (tzn. má hodnotu `NULL`), dojde k přiřazení hodnoty z proměnné cílového uzlu do proměnné zdrojového uzlu. Stejně jako při vytvoření NPC se následně volá funkce `TryToFindNextNode()` a celý proces cesty k cíli se opakuje. Pokud má uzel akci s hodnotou `StartExit`, dochází k nastavení aktuálního módu na End. V rámci tohoto módu nastává změna viditelnosti NPC a jeho následná deaktivace, po níž je NPC vloženo do poolu uvnitř instance třídy `AIManager`.

Třetím módem je takzvaný HelpMode. Tento mód se aktivuje, jakmile instanční proměnná `tolerance` typu `ParametrRange` dosáhne maximální hodnoty. Tato proměnná je ovlivněna hráčovým gestem žebrání a hlasovými příkazy. Pokud se hráč nachází zorném úhlu NPC, provádí gesto žebrání a zároveň jej od NPC dělí definovaná vzdálenost, tak se průběžně navyšuje hodnota proměnné `tolerance` v závislosti na aktuální úrovni zápachu hráče. Jakmile některá z těchto podmínek neplatí, hodnota této proměnné konstantně klesá. Pro rozpoznávání hlasových příkazů je instance třídy `NPCBehavior` přihlášena k odběru události `OnVoiceHypothesis` ze třídy `PlayerVoice`. Při vyvolání této události se porovnává textový přepis rozpoznaného zvuku s hlasovými příkazy určenými pro žebrání. Pokud je nalezena shoda a NPC je v dostatečné vzdálenosti od hráče, pak dochází ke zvýšení hodnoty tolerance. NPC se v průběhu HelpMode pohybuje místo k cílovému uzlu směrem k hráči. Jakmile se NPC nachází v těsné blízkosti hráče, dochází k přehrání animace představující odevzdání předmětu hráči. Po jejím dokončení se objekt vrací zpět do Basic módu.

O animace a 3D model se stará třída `NPCModel`. Instance této třídy je vázána na objekt `NPCBehavior`. Viditelnost je dána proměnnou `transparency` typu float. Průběžně se provádí kontrola, zdali se tato hodnota rovná cílové hodnotě obsažené v proměnné `targetTransparency`. Při nerovnosti dochází k plynulé úpravě viditelnosti na požadovanou hodnotu. Děje se tak například při opuštění viditelné zóny nebo v rámci End módu. Pro změnu viditelnosti při vykreslování

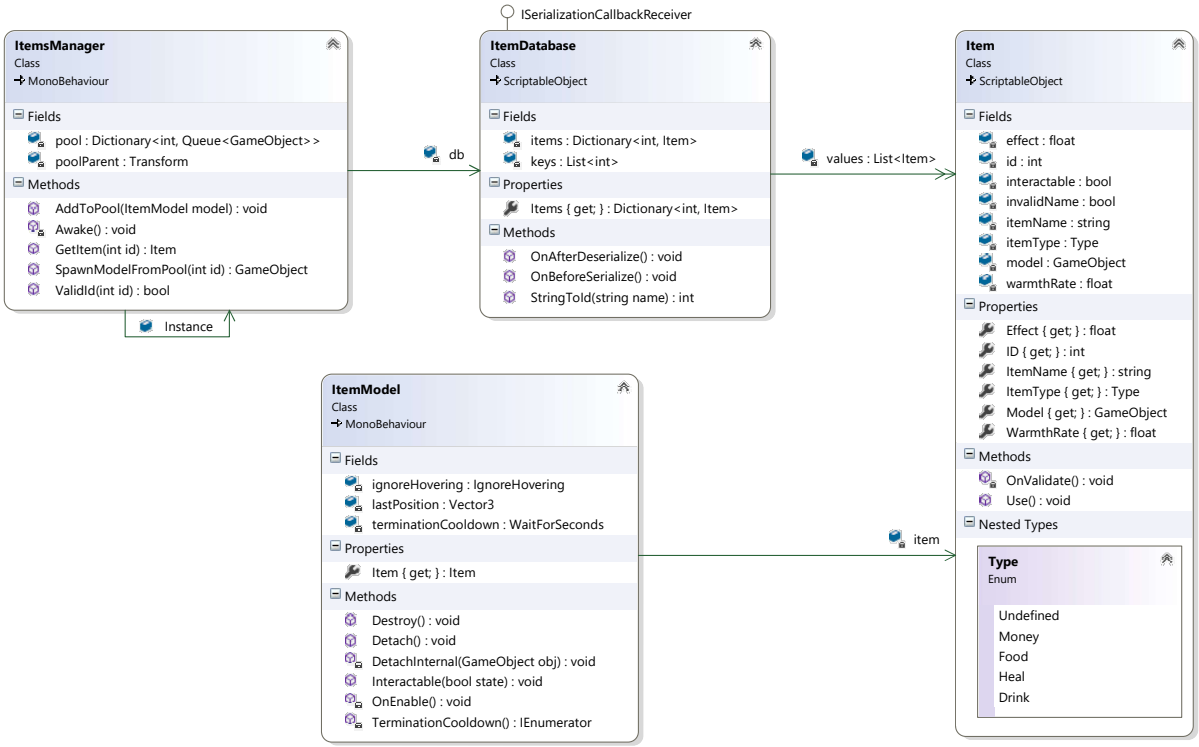
modelu byl vytvořený shader `Standard Crossfade`. Tento shader využívá difúzní a spekulární složku a je napojený na Unity osvětlovací model. V rámci fragment shaderu je pro právě zpracovávaný pixel spočítána jeho pozice v prostoru obrazovky. Tato hodnota je předaná do funkce UnityApplyDitherCrossFade (předdefinovaná funkce enginu obsažená v souboru `UnityCG.cginc`). Tato funkce pro danou pozici provede takzvaný dithering, jehož výsledkem je případné zahození pixelu v závislosti na aktuální hodnotě v parametru `unity_LODFade`. Hodnota tohoto parametru se rovná hodnotě `transparency` v instanci třídy `NPCModel`. Animace jsou realizovány pomocí třídy `Animator` (třída v Unity API sloužící pro práci s animacemi), jež je propojena s instancí třídy `NPCModel`. Pro NPC se využívá animace chůze[11] a animace odevzdání předmětu[12]. Animace chůze se přehrává ve smyčce při pohybu NPC. K zahájení přehrávání animace odevzdání předmětu dochází, jak již bylo zmíněno dříve, v HelpModu v rámci `NPCBehavior`. Animace před svým koncem vyvolává událost s názvem `AnimationEvent` s parametrem `animation` typu string o textové hodnotě `give`. K této události je přihlášena stejnojmenná funkce v `NPCBehavior`. Pokud se parametr `animation` rovná hodnotě `give`, pak se na pozici ruky modelu NPC vytvoří náhodný předmět z kolekce určené v konfiguračním souboru.

4.6 Předměty

Tato podkapitola se věnuje herním předmětům. V předloze jsou herní předměty zobrazovány v rámci 2D GUI obrazovek (například inventář, nakupování atd.) pomocí 2D obrázků. Všechny 2D obrazovky a mechaniky, které využívají předměty, jsou ve VR v rámci této práce přeneseny do 3D. Je tedy nutné přenést do prostoru i samotné předměty. Pro dosažení tohoto požadavku a zajištění jednoduché práce s předměty byly vytvořeny následující třídy: `Item`, `ItemModel`, `ItemDatabase` a `ItemManager` (viz obrázek č. 22).

Pro zajištění interakce hráče s předměty v prostoru pomocí ovladačů se využívají třídy z takzvaného InteractionSystem. Jedná se o třídy zahrnuté uvnitř SteamVR Pluginu pro engine Unity. InteractionSystem řeší čtení vstupů z ovladačů, nástroje pro detekci kolize ovladačů s kolizními objekty v Unity a spoustu dalších pomocných nástrojů, které jsou velice užitečné pro řešení interakce hráče s herním světem, respektive s objekty v něm obsaženými.

[11] https://www.assetstore.unity3d.com/en/content/5330
[12] https://www.mixamo.com/



Obrázek 22: Třídní diagram herních předmětů

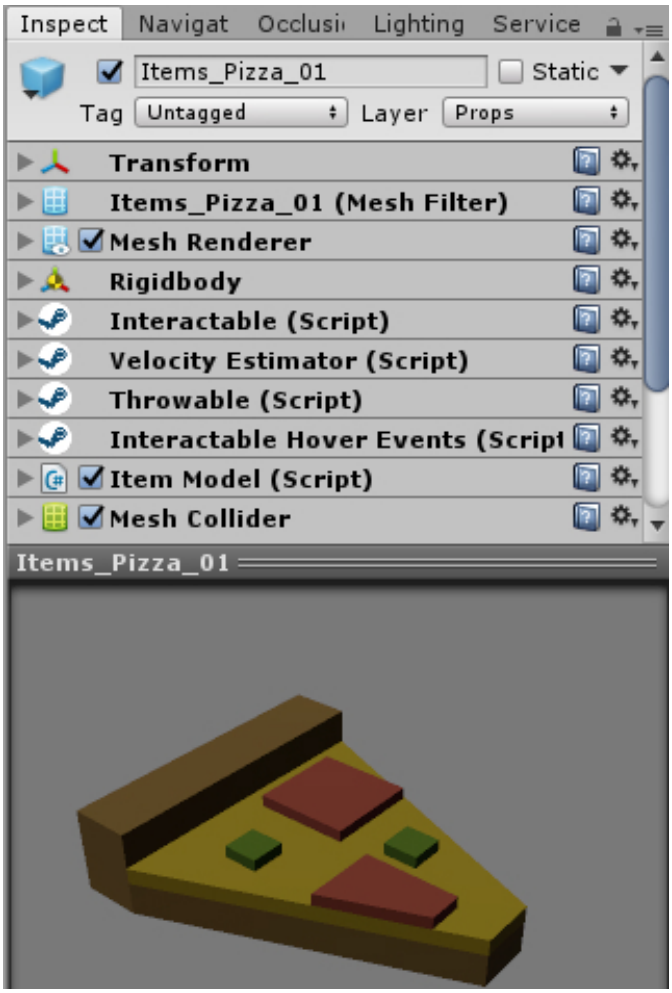
4.6.1 Třída Item

Třída `Item` zastřešuje veškeré vlastnosti a specifika předmětů. Předmět je definován názvem, typem předmětu (proměnná `itemType` vnořeného typu `Type`, která může nabývat hodnot `Food`, `Drink` atd.), hořlavostí (která se využívá při vhození předmětu do ohně), úrovní efektu na hráče (například v případě jídla se jedná o hodnotu určující, nakolik předmět hráče zasytí) a vazbou na objekt představující 3D model pro tento předmět pod proměnnou `model`. Název instance třídy `Item` musí být jedinečný. Na základě jeho hodnoty se generuje unikátní identifikační číslo, které se ukládá do instanční proměnné `id` typu int. Tato hodnota se používá například pro vyhodnocení, zdali se jedná o stejný předmět (porovnáním jejich id). Třída `Item` dědí ze třídy `ScriptableObject`. V prostředí Unity tak lze vytvářet instance této třídy formou serializovaných souborů. Díky tomuto přístupu můžeme hodnoty jednotlivých proměnných předmětu editovat prostřednictvím inspectoru.

4.6.2 Interaktivní předměty

Pro práci s interaktivním 3D předmětem vznikla třída `ItemModel`. Ta obsahuje vazbu na entitu typu `Item`. Díky tomuto spojení lze přes ItemModel přistupovat k vlastnostem definovaných v instanci třídy`Item`. Instance `ItemModel` společně s instancemi tříd z Unity API a InteractionSystem typu `MeshRenderer` (sloužící k vykreslování 3D modelu), `Rigidbody` (starající se například

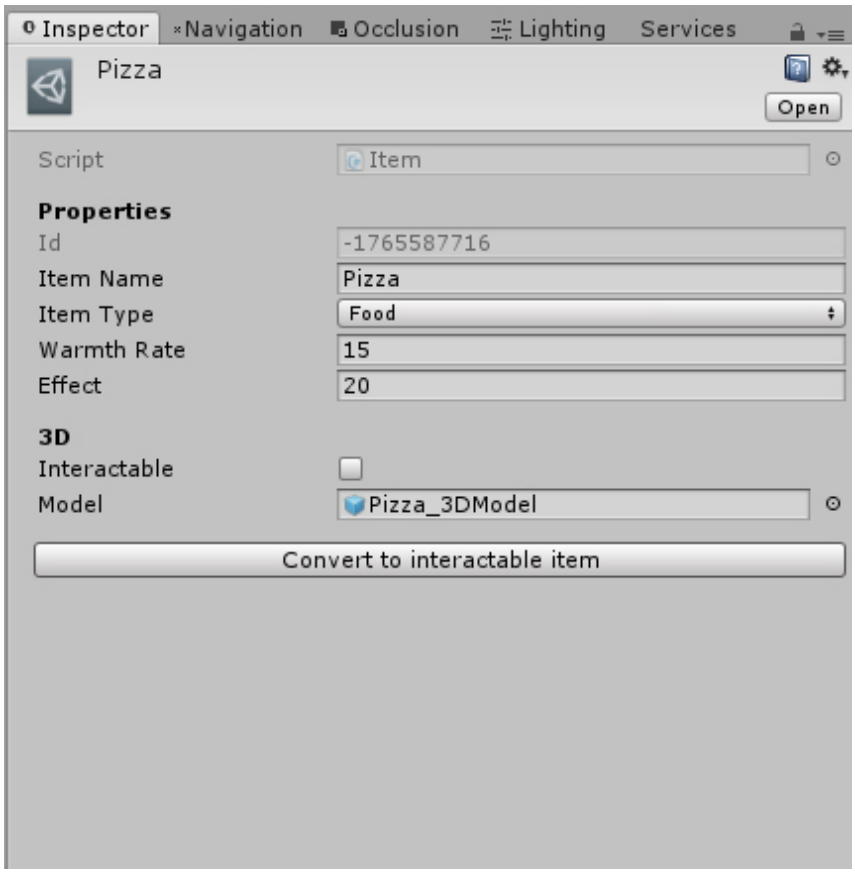
o působení gravitace), `Collider` (definující kolizní zónu), `Interactable`, `VelocityEstimator`, `Throwable` a `InteractableHoverEvents` tvoří dohromady interaktivní předmět (viz obrázek č. 23). Poslední čtyři jmenované třídy pro InteractionSystem určují, že se jedná o interakční objekt, se kterým lze házet a který lze brát do ruky po najetí ovladačem. Díky kombinaci těchto tříd může hráč s předmětem přirozeně interagovat. Kromě vazby na třídu `Item` slouží `ItemModel` také k automatickému mazání již nevyužívaných předmětů. V rámci této třídy tedy v pravidelných časových intervalech dochází k vyhodnocení, zdali se od poslední kontroly změnila pozice předmětu. Pokud žádná změna nenastala a předmět se nenachází poblíž hráče, dojde k zániku tohoto předmětu z herního světa.



Obrázek 23: Ukázka složení interaktivního předmětu

4.6.2.1 Třída ItemEditor Vytváření interaktivních předmětů je v prostředí Unity Editoru zdlouhavý a rutinní proces. U každého předmětu je totiž třeba přiřadit všechny instance, provést jejich nastavení, přidat vazby atd. Pro ušetření práce a času byl vytvořený nástroj, který se automaticky stará o veškerá nastavení. Tento nástroj se používá za pomoci třídy `ItemEditor`,

jež upravuje grafickou vizualizaci instancí třídy `Item`. Kromě standardního grafického zpracování proměnných určených k editaci v rámci Unity Editoru se zabývá objektem v proměnné `model` v aktuálně vybrané entitě třídy `Item`. Nad tímto objektem provádí sérii kontrol, které zjišťují, zda tento objekt obsahuje vše potřebné pro interagování. Pokud se ukáže, že nejsou splněny všechny náležitosti, v rámci inspectoru se vykreslí tlačítko pro automatickou úpravu objektu do podoby interaktivního předmětu (viz obrázek č. 24). Po kliknutí proběhne automatická konverze, jejímž výsledkem je interaktivní předmět.



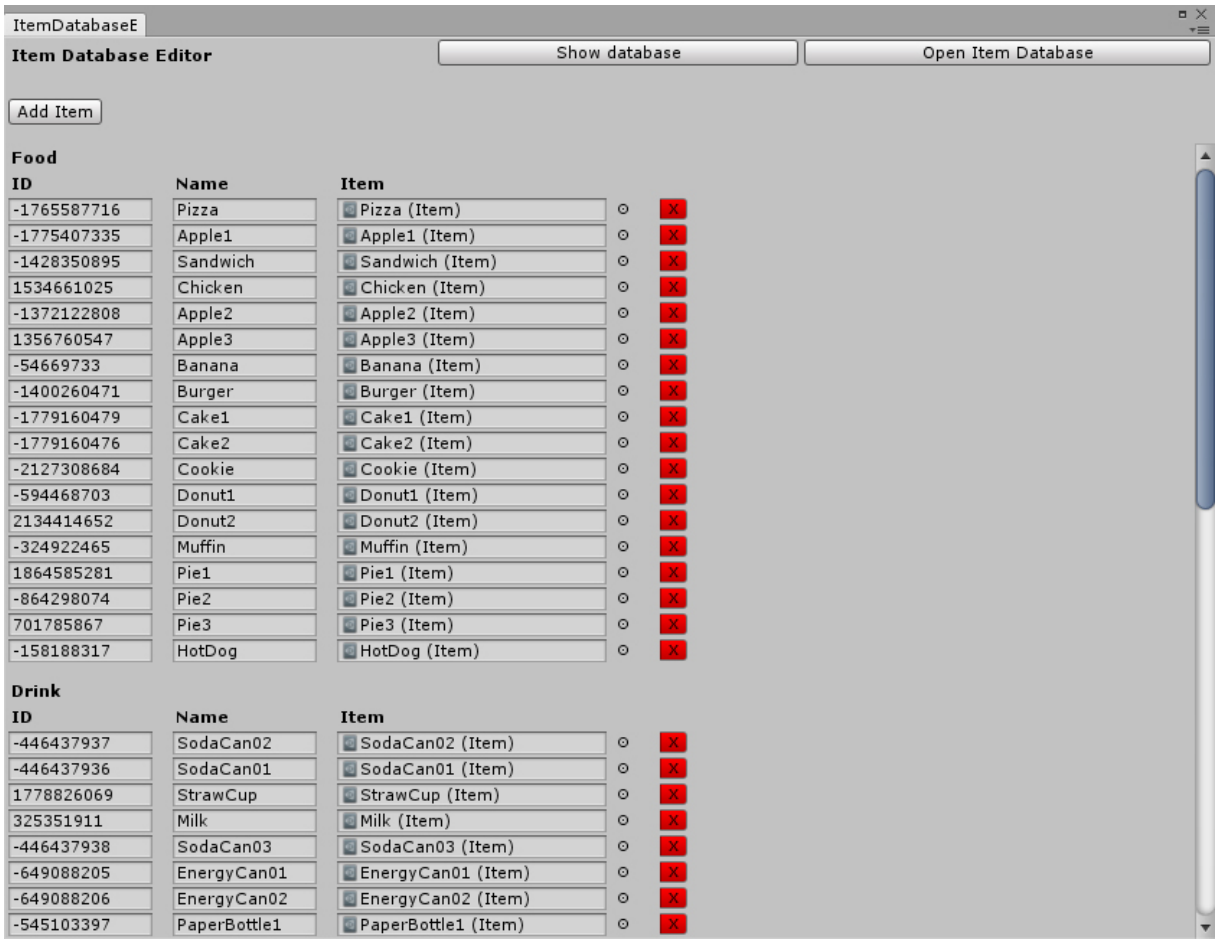
Obrázek 24: Ukázka přizpůsobého zobrazení itemu v Inspectoru

4.6.3 Třída ItemDatabase

Třída `ItemDatabase` slouží jako databáze všech předmětů ve hře. Je realizovaná formou kolekce typu Dictionary s názvem `items`. Hodnoty této kolekce jsou reprezentovány konkrétními instancemi třídy `Item`. Klíčem k těmto hodnotám jsou pak jejich identifikační čísla. Tato třída dědí ze `ScriptableObject`. Obsah `ItemDatabase` je tak serializovaný do souboru. Unity však v základní verzi nepodporuje serializaci třídy `Dictionary`. Řešením je tedy vlastní serializační logika, při které kolekci `items` převedeme do serializovatelné podoby. Z tohoto důvodu třída `ItemDatabase` implementuje rozhraní `ISerializationCallbackReceiver`. Při serializaci a deserializaci Unity volá funkce `OnBeforeSerialize()` a `OnAfterDeserialize()` nad objektem

implementujícím toto rozhraní. V rámci těchto funkcí lze vytvořit vlastní serializační logiku. Aby bylo možné použít Dictionary, byla třída `ItemDatabase` rozšířena o dvě kolekce typu List. První z nich reprezentuje klíče, druhá hodnoty kolekce `items`. Při serializaci jsou tyto kolekce naplněny obsahem z kolekce předmětů. Při deserializaci se z nich naopak vytváří obsah v proměnné `items`.

4.6.3.1 Třída ItemDatabaseEditor S cílem jednoduchého vkládání a editace instancí třídy `Item` v databázi předmětů vznikla třída `ItemDatabaseEditor` dědící z třídy `EditorWindow` (interní třída obsažená v UnityEditor API sloužící k vykreslování samostatných oken uvnitř prostředí Unity Editoru). Tato třída databázi vizualizuje formou separátního okna. Databáze se v rámci tohoto okna, respektive instance `ItemDatabaseEditor`, zobrazuje pomocí 2D GUI (viz obrázek č. 25). `ItemDatabaseEditor` prochází obsah kolekce předmětů, přičemž jednotlivé hodnoty uložené v databázi zobrazuje v rámci scrollovatelného seznamu řazeného podle typu předmětu. Pomocí tlačítka u jednotlivých hodnot je realizováno mazání z databáze. Případné chyby vzniklé při přidáváním či editací (neplatný název, nepřiřazená hodnota předmětu atd.) jsou okamžitě vizualizovány zvýrazněním chybného místa.



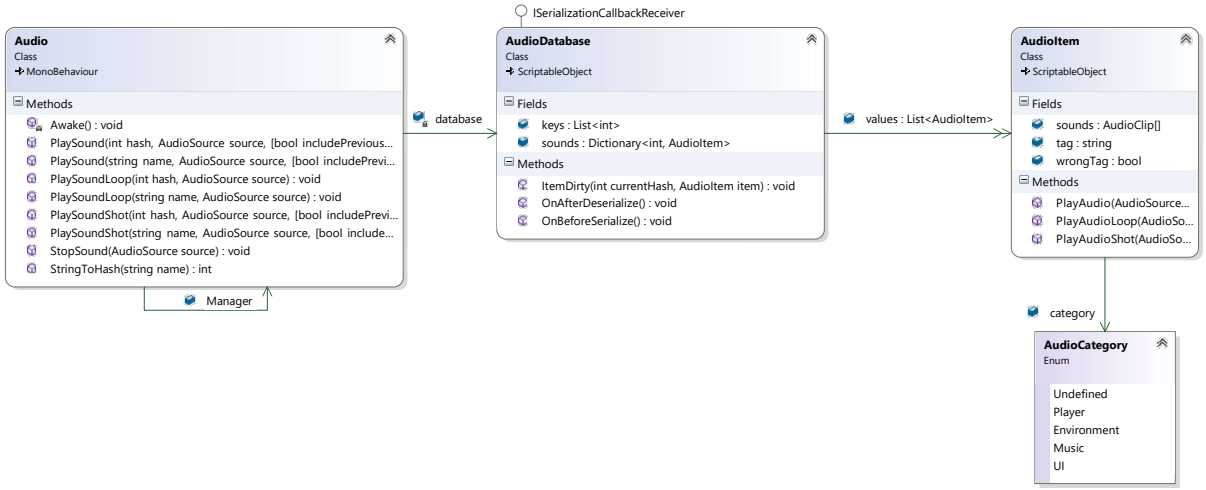
Obrázek 25: Ukázka ItemDatabase

4.6.4 Třída ItemsManager

Pro vytváření interaktivních předmětů za chodu hry byla zhotovena třída `ItemsManager`. Obsahuje vazbu na databázi předmětů. Předměty jsou vytvářeny pomocí návrhového vzoru Object Pool, který na rozdíl od poolu v rámci třídy `AIManager` není omezen velikostně. Pokud si nějaký objekt vyžádá předmět, nejprve se prohledá pool, aby se zjistilo, zdali takový předmět není k dispozici. Jestliže se daný předmět v poolu nenachází, dochází následně k jeho vytvoření a inicializování.

4.7 Zvuk

V Unity API je zabudovaná podpora pro přehrávání zvuků prostřednictvím instancí třídy `AudioSource`. Tyto instance přehrávají přiřazené zvukové soubory. Tento systém ovšem neumožňuje vybírat zvuky náhodně. Každý objekt, který chce přehrát zvuk, v sobě musí obsahovat vazby na zvukové soubory, aby je mohl předat k přehrání atd. Pro zjednodušení práce se zvukem byly vytvořeny třídy `AudioItem`, `AudioDatabase` a `Audio` (viz obrázek č. 26). `AudioItem` slouží pro referencování zvukových souborů. Může najednou obsahovat jeden či více zvuků, z kterých se jeden náhodně pro přehrávání vybírá. Lze tedy jednoduše docílit větší variability pro určitý typ zvuku, jelikož se nepřehrává pokaždé tentýž zvuk, ale vybírá se z několika jeho variant. Podobným způsobem, jakým je realizovaná databáze předmětů, je zpracována i audio databáze. Ta obsahuje kolekci jednotlivých entit třídy `AudioItem`. Práci se zvukem řídí třída `Audio`. V instanci této třídy je vazbu na databázi a vychází z návrhového vzoru Singleton. Je tak umožněn globální přistup k přehrávání zvuků.



Obrázek 26: Třídní diagram pro práci se zvukem

4.8 Herní mechaniky

Tato podkapitola se zabývá realizací herních mechanik (stanovených při návrhu v kapitole 3.2), kterým se práce v předchozích částech dosud nevěnovala. Konkrétně se tedy zabývá vybíráním popelnic, snižováním zápachu, nakupováním a počasím.

4.8.1 Vybírání popelnic

V rámci herní mapy jsou rozmístěny desítky popelnic a odpadkových košů (reprezentovány 3D modelem a instancí třídy `Bin`). Jsou rozlišovány podle velikosti na malé, střední a velké. V konfiguračním souboru jsou pro každou velikost nastaveny konkrétní předměty a jejich množství. V malých odpadkových koších, hráč může nalézt vždy maximálně jeden předmět drobnějších rozměrů (například zbytky jídla atp.). V těch největších popelnicích lze nalézt až deset předmětů najednou (od jídla až po staré krabice, nefunkční elektronická zařízení atd.). Obsah dané popelnice se náhodně generuje (viz obrázek č. 27). K vygenerování obsahu dochází ve chvíli, kdy vzdálenost mezi hráčem a konkrétní popelnicí klesne pod stanovenou hranici. Každá popelnice má definovanou kolizní zónu kopírující proporce jejího modelu. Tato zóna detekuje, zda se tělo hráče nachází v prostoru dané popelnice. Pakliže tato situace nastane, narůstá aktuální úroveň zápachu hráče.

Obrázek 27: Ukázka vygenerovaných předmětů v popelnici

4.8.2 Snižování zápachu

Pro snížení momentální hodnoty zápachu jsou ve hře použity vodní fontány. Ty jsou tvořeny 3D modelem a tryskající vodou. Voda je realizovaná pomocí particle systému. Aby tekoucí voda vypadala věrohodně, využívá při vykreslování šumovou texturu, která způsobuje turbulence při pohybu jednotlivých částic. V rámci fontány jsou rozmístěny kolizní zóny zjednodušeně kopírující tvar modelu (viz obrázek č. 28). Jednotlivé částice vody jsou rovněž napojeny na kolizní systém, přičemž při dopadu na některou z kolizních zón fontány vzniká efekt rozprsknutí vody. Tento

stejný efekt je použit i při kolizi vody s hráčem, při níž navíc dochází k postupnému snižování parametru zápachu.

Obrázek 28: Kolizní zóny vodní fontány

4.8.3 Ohřívání u barelu

Mechanika ohřívání je tvořena 3D modelem barelu, efektem ohně a třídou `FirePlace`. V rámci vnitřního prostoru modelu barelu je vytvořena kolizní zóna. Při vhození herního předmětu do této oblasti dochází k jeho zničení a doba hoření v instanci třídy `FirePlace` je navýšena o hodnotu hořlavosti definované ve vhazovaném předmětu. Po dobu, kdy aktuální doba hoření přesahuje hodnotu 0, je barel, respektive instance třídy `FirePlace`, v takzvaném stavu hoření. Hráč, který se nachází v dosahu takto hořícího barelu, je zahříván (snižuje se hodnota parametru prochladnutí). Tento stav je vizualizovaný efektem ohně (viz obrázek č. 29), jenž je tvořen particle systémem a slabě svítícím bodovým světlem.

Obrázek 29: Vizualizace hořícího barelu

4.8.4 Nakupování

V rámci mapy se vyskytují různě dimenzované budovy. Je-li jejich obsah stejný a liší se pouze tvarem uvnitř, je zbytečné pro každou z nich vytvářet unikátně tvarované vnitřní prostory. Nakupování tedy probíhá prostřednictvím pro všechny budovy stejného 3D obchodu, který se nachází mimo herní mapu města. Do obchodu se hráč dostává pomocí teleportálů, které jsou umístěné v rámci herní mapy u vchodu vybraných budov. Aby hráč po čase stráveném v obchodě neztratil přehled, kde se v herním světě nachází, na dveřích ven jsou průhledná okna, skrz které lze vidět venkovní svět. Tohoto efektu je docíleno pomocí takzvaných reflection probes. Ty slouží pro zachycení okolního světa ve všech směrech. Objekty s reflektivním materiálem (v tomto případě okna u dveří) pak takto zachycený svět při svém vykreslení dokáží zrcadlit [38]. Unity při vykreslování těchto odrazů pro vyhodnocení, kterou část z reflection probe pro daný objekt zrcadlit, používá vždy aktuální natočení samotného objektu. Vstupy do obchodů se však nacházejí pod různým úhlem. Na oknech by se tak mohla zrcadlit jiná strana, než je potřeba. Pro vyřešení tohoto problému byly vytvořeny čtyři identické obchody, každý z nich je však natočený pod jiným úhlem (0°, 90°, 180°a 270°). Na základě vchodu a jeho umístění se používá jedna z těchto čtyřech variant, tak aby správně sedělo zrcadlení.

Obrázek 30: Ukázka realizace obchodu

Předměty, které si hráč může nakoupit, jsou umístěny v regálech. Nachází se u nich cedulka s názvem zboží a cenou (viz obrázek č. 30). Pro zvolení zboží se používají hlasové povely. Ty se skládají ze zahajovací fráze (například *I would like to buy*, *give me* atd.) a specifikování požadovaného množství a zboží. Systém na zpracování této fráze není omezen pouze na jedno slovo pro jeden produkt. Umožnuje používání odlišných slov a více slovních spojení pro označení identického zboží. Lze tak například říct *give me one milk*, ale také *give me four milk bottles*. Po zpracování požadavku od hráče vyskládá přítomný prodavač požadované zboží na pult. Aktuální cena za nákup se zobrazuje na terminálu. Zaplacení se provádí vyskládáním bankovek v hodnotě nákupu na pult. Po provedení transakce si hráč může vyskládané zboží z pultu vzít. Pokud

se hráč v průběhu nakupování rozhodne část nezaplaceného a již vyskládaného zboží změnit (například místo dvou lékárniček chce nyní jenom jednu), může tak učinit pomocí odstraňovací fráze. Ta je uvozena slovem *remove*, po kterém následuje specifikování množství a zboží, které chceme odebrat. Aby tato fráze nepůsobila strojově, umožňuje používání zájmen. Pro odstranění tak lze například použít povel *remove those two energy drinks*.

4.8.5 Počasí

Změnu počasí řídí třída `WeatherManager`. K úpravě počasí ve hře dochází v pravidelných intervalech, které jsou definovány v konfiguračním souboru. Konkrétně se jedná o změny mezi slunečným počasím a deštěm. Během srážek v okolí hráče prší (efekt realizovaný prostřednictvím particle systému), změnou barvy se obyčejné bílé mraky mění na bouřkové mraky a dochází ke snížení intenzity směrového světla a ztmavení celé oblohy (viz obrázek č. 31).

Obrázek 31: Srovnání vizualizace před deštěm a za deště

5 Testování

Tato kapitola se věnuje testování a ladění, které se průběžně provádělo při vývoji této práce. Funkčnost hry se ověřovala průběžně. Testování probíhalo na následující sestavě:

Tabulka 1: Použitá sestava při testování

Parametr	Hodnota
VR brýle	HTC Vive
CPU	AMD Ryzen 1600@3.9Ghz
GPU	Nvidia GTX 1060 6GB
RAM	16GB DDR4
HDD	Seagate 1TB

V enginu Unity jsou zabudované velmi kvalitní nástroje určené k ladění a testování v něm vyvíjených her. Od roku 2015 je v enginu zabudovaná podpora Visual Studia umožňující ladění veškerého kódu a knihoven dané aplikace (včetně využívání breakpointů, možnosti kontrolovat a upravovat proměnné a argumenty aj.), ať už v rámci editoru nebo v sestavené samostatně spustitelné aplikaci [39]. V Unity API je obsažena třída `Debug` sloužící k vypisování logovacích a chybových zpráv do konzole v prostředí editoru, respektive do logovacího souboru, pokud se jedná o již sestavenou aplikaci. Tyto hojně používané nástroje pro testování se při vývoji pro virtuální realitu ukázaly jako nedostatečné. Pro ověření funkčnosti je totiž často nutné, aby se testovalo přímo ve VR, kde však nemáme přístup k logovací konzoli. Jediná možnost je sundat VR brýle a podívat se na obsah logu v počítači. Používání breakpointu je rovněž velmi nepraktické. Běžně při najetí na breakpoint v kódu stačí danou část odkrokovat a následně pokračovat v testování. Avšak ve VR je tento proces navýšen o dobu, kterou vývojář tráví neustálým nasazováním a sundáváním VR brýlí. Při ověřování funkčnosti se vývojář navíc často vyskytuje někde v prostoru a kromě nasazování brýlí je doba ladění prodloužena o přesun k počítači a následný návrat do prostoru VR. Pro zlepšení efektivity testování se tak pro ověřování funkčnosti místo breakpointů a podobných technik využívaly hlavně grafické prvky zobrazované přímo ve VR brýlích, například na testování umělé inteligence se používaly koule umístěné nad modely jednotlivých NPC. Barva těchto koulí reprezentovala momentální chování NPC (viz obrázek č. 32), a bylo tedy možné snadno otestovat zda vše funguje, jak má, i v rámci VR.

Obrázek 32: Ukázka testování chování NPC za pomoci grafických prvků

Aby se kód určený pro testování a ladění nevyskytoval v sestavené aplikaci určené jen pro hraní, používají se direktivy preprocesoru. Díky těmto direktivám, respektive díky použití `if-endif` bloku pro danou direktivu, lze mít v kódu testovací funkce a příkazy, které se ve zkompilovaném kódu nemusí nacházet.

6 Závěr

Přepracování počítačové předlohy Hobo: Tough Life do virtuální reality ukázalo na velký potenciál této technologie, odkrylo některá její specifika, ale i slabé stránky. Největší slabinou jsou problémy plynoucí z technických omezení stávajících zařízení pro virtuální realitu. Ze zpracování rovněž vyplynulo, že běžně používané postupy designování a vývoje klasických počítačových her jsou v případě virtuální reality často neproveditelné (například nelze využívat 2D GUI). Převedení si tedy vyžádalo vytvoření mnoha alternativních návrhů pro přenesení herních mechanik z předlohy do prostředí virtuálního světa.

Přínosem této bakalářské práce pro mě bylo prohloubení znalosti vytváření her a získání zkušeností s technologií virtuální reality i vývojem her v tomto prostředí. V návaznosti na tuto práci by bylo možné pokračovat v rozšiřování interakce s herním světem a portování na jiné platformy, jako například na mobilní virtuální realitu.

Jiří Vašica

Literatura

[1] The virtual reality industry can't stop growing — but supply of workers is limited. *CNBC* [online]. 2017 [cit. 2018-04-08]. Dostupné z: https://www.cnbc.com/2017/12/08/virtual-reality-continues-to-grow--but-supply-of-workers-is-limited.html

[2] Virtual Reality Market Size Growth & Analysis, VR Industry Report 2025. *Grand view researcher* [online]. 2017 [cit. 2018-04-08]. Dostupné z: https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-market

[3] Applications Of Virtual Reality. *Virtual Reality Society* [online]. 2017 [cit. 2018-04-08]. Dostupné z: https://www.vrs.org.uk/virtual-reality-applications/

[4] What is Virtual Reality? [Definition and Examples] *Marxent labs* [online]. 2015 [cit. 2018-04-08]. Dostupné z: https://www.marxentlabs.com/what-is-virtual-reality/

[5] History Of Virtual Reality. *Virtual Reality Society* [online]. 2017 [cit. 2018-04-08]. Dostupné z: https://www.vrs.org.uk/virtual-reality/history.html

[6] CES 2018: The 5 Biggest VR Tech Updates. *UploadVR* [online]. 2018 [cit. 2018-04-08]. Dostupné z: https://uploadvr.com/best-of-ces-2018-5-big-updates/

[7] Advanced VR Rendering. *Valve* [online]. 2015 [cit. 2018-04-08]. Dostupné z: http://media.steampowered.com/apps/valve/2015/Alex_Vlachos_Advanced_VR_Rendering_GDC2015.pdf

[8] The very real health dangers of virtual reality. *CNN* [online]. 2017 [cit. 2018-04-08]. Dostupné z: https://edition.cnn.com/2017/12/13/health/virtual-reality-vr-dangers-safety/index.html

[9] Why does virtual reality make you want to puke? *The Daily Dot* [online]. 2016 [cit. 2018-04-08]. Dostupné z: https://www.dailydot.com/parsec/virtual-reality-sickness-science/

[10] VR Needs To Hit 16K To Match Retinal Resolution. *VRFocus* [online]. 2015 [cit. 2018-04-08]. Dostupné z: https://www.vrfocus.com/2015/03/abrash-vr-needs-hit-16k-match-retinal-resolution/

[11] What is the "screen-door effect"and why does it happen? *VRHeads* [online]. 2016 [cit. 2018-04-08]. Dostupné z: https://www.vrheads.com/what-screen-door-effect-and-why-does-it-happen

[12] How long in VR is too long? *VRHeads* [online]. 2016 [cit. 2018-04-08]. Dostupné z: https://www.vrheads.com/how-long-vr-too-long

[13] Virtual Reality Still Has 5 Big Problems to Overcome. *Make Use Of* [online]. 2016 [cit. 2018-04-08]. Dostupné z: https://www.makeuseof.com/tag/virtual-reality-still-5-big-problems-overcome/

[14] Long-term effects of virtual reality use need more research, say scientists. *The Guardian* [online]. 2016 [cit. 2018-04-08]. Dostupné z: https://www.theguardian.com/technology/2016/mar/19/long-term-effects-of-virtual-reality-use-need-more-research-say-scientists

[15] Hobo: Tough Life *Steam* [online]. 2018 [cit. 2018-04-08]. Dostupné z: http://store.steampowered.com/app/632300/Hobo_Tough_Life/

[16] Optimizations *Unity documentation* [online]. 2018 [cit. 2018-04-08]. Dostupné z: https://docs.unity3d.com/Manual/MobileOptimisation.html

[17] Secrets to Creating Low Poly Illustrations in Blender *Envato tutorials* [online]. 2014 [cit. 2018-04-08]. Dostupné z: https://cgi.tutsplus.com/tutorials/secrets-to-creating-low-poly-illustrations-in-blender--cg-31770

[18] Role-Playing Game (RPG) *Technopedia* [online]. 2016 [cit. 2018-04-13]. Dostupné z: https://www.techopedia.com/definition/27052/role-playing-game-rpg

[19] Virtual Selection: The Rise Of The Survival Game *IGN* [online]. 2013 [cit. 2018-04-13]. Dostupné z: http://www.ign.com/articles/2013/07/05/virtual-selection-the-rise-of-the-survival-game

[20] Medical Definition of Borborygmi *MedicineNet* [online]. 2016 [cit. 2018-04-18]. Dostupné z: https://www.medicinenet.com/script/main/art.asp?articlekey=25872

[21] Unity for VR and AR *Unity3D* [online]. 2018 [cit. 2018-04-08]. Dostupné z: https://unity3d.com/unity/features/multiplatform/vr-ar

[22] A feature-rich and highly flexible editor *Unity3D* [online]. 2018 [cit. 2018-04-14]. Dostupné z: https://unity3d.com/unity/editor

[23] Rendering Paths *Unity documentation* [online]. 2018 [cit. 2018-04-08]. Dostupné z: https://docs.unity3d.com/Manual/RenderingPaths.html

[24] Single-Pass Stereo rendering *Unity documentation* [online]. 2018 [cit. 2018-04-08]. Dostupné z: https://docs.unity3d.com/Manual/SinglePassStereoRendering.html

[25] Optimizing graphics performance *Unity documentation* [online]. 2018 [cit. 2018-04-14]. Dostupné z: https://docs.unity3d.com/Manual/MobileOptimisation.html

[26] Draw call batching *Unity documentation* [online]. 2018 [cit. 2018-04-14]. Dostupné z: https://docs.unity3d.com/Manual/DrawCallBatching.html

[27] Occlusion Culling *Unity documentation* [online]. 2018 [cit. 2018-04-14]. Dostupné z: https://docs.unity3d.com/Manual/OcclusionCulling.html

[28] GPU instancing *Unity documentation* [online]. 2018 [cit. 2018-04-14]. Dostupné z: https://docs.unity3d.com/Manual/GPUInstancing.html

[29] Level of Detail (LOD) *Unity documentation* [online]. 2018 [cit. 2018-04-14]. Dostupné z: https://docs.unity3d.com/Manual/LevelOfDetail.html

[30] Oculus Rift Vs. HTC Vive Vs. PlayStation VR *Tom's hardware* [online]. 2016 [cit. 2018-04-19]. Dostupné z: https://www.tomshardware.com/reviews/vive-rift-playstation-vr-comparison,4513-6.html

[31] KARLSSON, Rasmus a Alvar SVENINGE. *Virtual Reality Locomotion: Four Evaluated Locomotion Methods* [online]. Trollhättan, 2017 [cit. 2018-04-19]. Dostupné z: http://www.diva-portal.org/smash/get/diva2:1144090/FULLTEXT02.pdf. University College West, Department of Economics and IT.

[32] Navigation System in Unity *Unity documentation* [online]. 2018 [cit. 2018-04-19]. Dostupné z: https://docs.unity3d.com/Manual/nav-NavigationSystem.html

[33] Inverse Kinematics *Unity documentation* [online]. 2018 [cit. 2018-04-19]. Dostupné z: https://docs.unity3d.com/Manual/InverseKinematics.html

[34] What is a Particle System? *Unity documentation* [online]. 2018 [cit. 2018-04-20]. Dostupné z: https://docs.unity3d.com/Manual/PartSysWhatIs.html

[35] Object Pool Design Pattern *Source making* [online]. 2018 [cit. 2018-04-21]. Dostupné z: https://sourcemaking.com/design_patterns/object_pool

[36] ScriptableObject *Unity documentation* [online]. 2018 [cit. 2018-04-21]. Dostupné z: https://docs.unity3d.com/Manual/class-ScriptableObject.html

[37] NavMesh Agent *Unity documentation* [online]. 2018 [cit. 2018-04-22]. Dostupné z: https://docs.unity3d.com/Manual/class-NavMeshAgent.html

[38] Reflection Probes *Unity documentation* [online]. 2018 [cit. 2018-04-23]. Dostupné z: https://docs.unity3d.com/Manual/ReflectionProbes.html

[39] Overview of Visual Studio Tools for Unity *MSDN* [online]. 2018 [cit. 2018-04-24]. Dostupné z: https://msdn.microsoft.com/en-us/library/dn940026.aspx

A Obsah přiloženého DVD

Na přiloženém DVD se nacházejí tyto složky:

- **Text** – obsahuje text bakalářské práce ve formátu PDF/A,
- **Build** – výsledná samostatně spustitelná hra,
- **Unity projekt** – zdrojové soubory Unity projektu,
- **Video** – videozáznam zachycující aspekty výsledné aplikace.